US006308317B1

(12) United States Patent
Wilkinson et al.

(10) **Patent No.:** **US 6,308,317 B1**
(45) **Date of Patent:** **Oct. 23, 2001**

(54) **USING A HIGH LEVEL PROGRAMMING LANGUAGE WITH A MICROCONTROLLER**

(75) Inventors: **Timothy J. Wilkinson**, London (GB); **Scott B. Guthery**, Belmont, MA (US); **Ksheerabdhi Krishna**; **Michael A. Montgomery**, both of Cedar Park, TX (US)

(73) Assignee: **Schlumberger Technologies, Inc.**, Austin, TX (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **08/957,512**

(22) Filed: **Oct. 24, 1997**

Related U.S. Application Data

(60) Provisional application No. 60/029,057, filed on Oct. 25, 1996.

(51) **Int. Cl.**[7] **G06F 13/00**
(52) **U.S. Cl.** **717/5**
(58) **Field of Search** 395/705; 717/5

(56) **References Cited**

U.S. PATENT DOCUMENTS

4,256,955		3/1981	Giraud et al.	235/380
4,650,975	*	3/1987	Kitchener	235/375
4,777,355		10/1988	Takahira .	
4,797,543		1/1989	Watanabe	235/492
4,877,947		10/1989	Mori	235/381
5,064,999		11/1991	Okamoto et al.	235/379
5,195,130		3/1993	Weiss et al.	379/98
5,406,380		4/1995	Teter	358/332
5,500,517	*	3/1996	Cagliostro	235/486
5,537,474	*	7/1996	Brown et al.	380/23
5,544,086		8/1996	Davis et al.	364/408
5,550,919		8/1996	Kowalski	380/23
5,590,197	*	12/1996	Chen et al.	380/24
5,604,802		2/1997	Holloway	380/24
5,613,012		3/1997	Hoffman et al.	382/115
5,650,761		7/1997	Gomm et al.	235/381
5,689,565		11/1997	Spies et al.	380/25
5,692,132		11/1997	Hogan	395/227
5,734,150		3/1998	Brown et al.	235/381
5,742,756	*	4/1998	Dillaway et al.	713/200
5,761,306		6/1998	Lewis	380/21
5,768,419		6/1998	Gundlach et al.	395/187.01
5,811,771		9/1998	Dethloff	235/380
5,815,657	*	11/1998	Williams et al.	713/200
5,841,866	*	11/1998	Bruwer et al.	380/23
5,844,218	*	12/1998	Kawan et al.	235/380
5,889,941	*	3/1999	Tushie et al.	713/200
5,915,226	*	6/1999	Martineau	455/558
5,923,884	*	7/1999	Peyret et al.	395/712

FOREIGN PATENT DOCUMENTS

0 829 828 A1	3/1998	(EP) .
0889393 A2	1/1999	(EP) .
0 427 465 A2	5/1999	(EP) .
2 191 029 A	12/1987	(GB) .
2 261 973 A	6/1993	(GB) .
WO 98/19237	5/1998	(WO) .

OTHER PUBLICATIONS

Rodley, Writing Java Applets, Corolis Group Books, Chapter 1. Apr. 15, 1996.*

Kung et al., Developing an Object–Oriented Software Testing and Maintaincane Enviroment, Oct. 1995, pp. 75–87.*

Cheng et al., Securing the Internet Protocol, Sep. 1995, p. 257.*

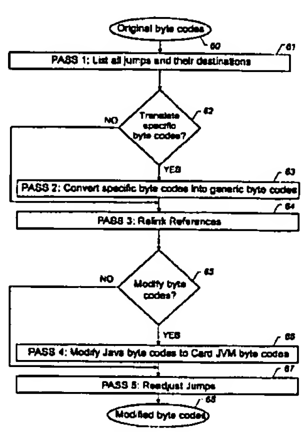
(List continued on next page.)

Primary Examiner—Mark R. Powell
Assistant Examiner—John Q. Chavis
(74) *Attorney, Agent, or Firm*—Pehr B. Jansson; Danita J. M. Maseles

(57) **ABSTRACT**

An integrated circuit card is used with a terminal. The integrated circuit card includes a memory that stores an interpreter and an application that has a high level programming language format. A processor of the card is configured to use the interpreter to interpret the application for execution and to use a communicator of the card to communicate with the terminal.

87 Claims, 23 Drawing Sheets



OTHER PUBLICATIONS

Sandhu et al., Authentication,Access Control, and Audit, Mar. 1996, pp. 241–243.*

Gosling, Audio/Video Sequence of Invited Presentations, May, 1996, pp. 1–4.*

Blundon, The Center of the universe is a database, Jul. 1996, pp. 1–5.*

Wingfield et al., News: Java Brews Trouble for Microsoft, Nov. 1995, pp. 1–2.*

0 427 465 A3—Examiner's search report for 0 427 465 A2 May 15, 1991.

0 356 237 A3—Examiner's search report for 0 356 237 A2 Feb.28, 1990.

PCT International Search Report dated Mar. 15, 1999.

PCT International Search Report, Dec. 29,1998, 7 pages.

PCT/US97/1899—Search Report.

* cited by examiner

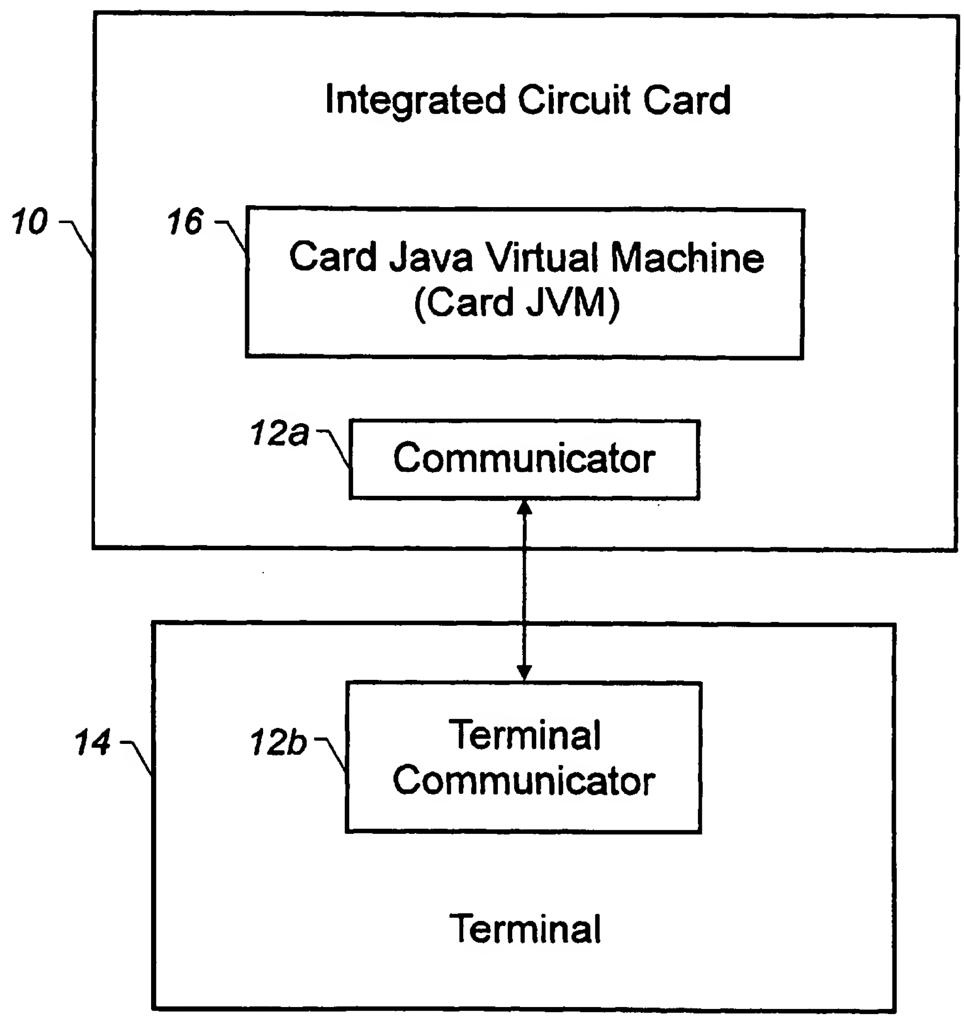

Integrated Circuit Card
10
16
Card Java Virtual Machine
(Card JVM)
12a
Communicator
14
12b
Terminal
Communicator
Terminal

FIGURE 1

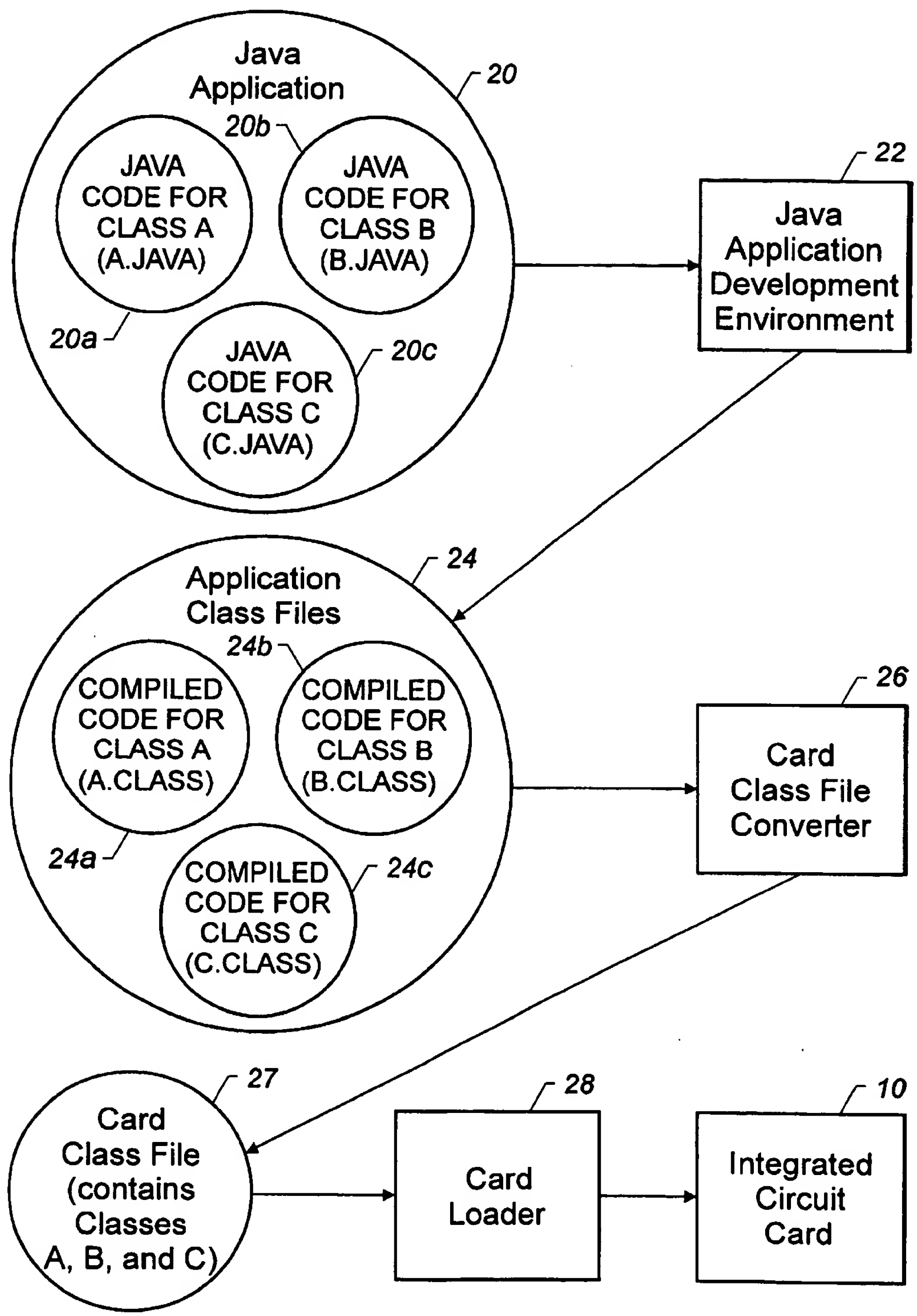

Java
Application
20
20b
JAVA
CODE FOR
CLASS A
(A.JAVA)
JAVA
CODE FOR
CLASS B
(B.JAVA)
20a
JAVA
CODE FOR
CLASS C
(C.JAVA)
20c
22
Java
Application
Development
Environment
Application
Class Files
24
24b
COMPILED
CODE FOR
CLASS A
(A.CLASS)
COMPILED
CODE FOR
CLASS B
(B.CLASS)
24a
COMPILED
CODE FOR
CLASS C
(C.CLASS)
24c
26
Card
Class File
Converter
27
Card
Class File
(contains
Classes
A, B, and C)
28
Card
Loader
10
Integrated
Circuit
Card

FIGURE 2

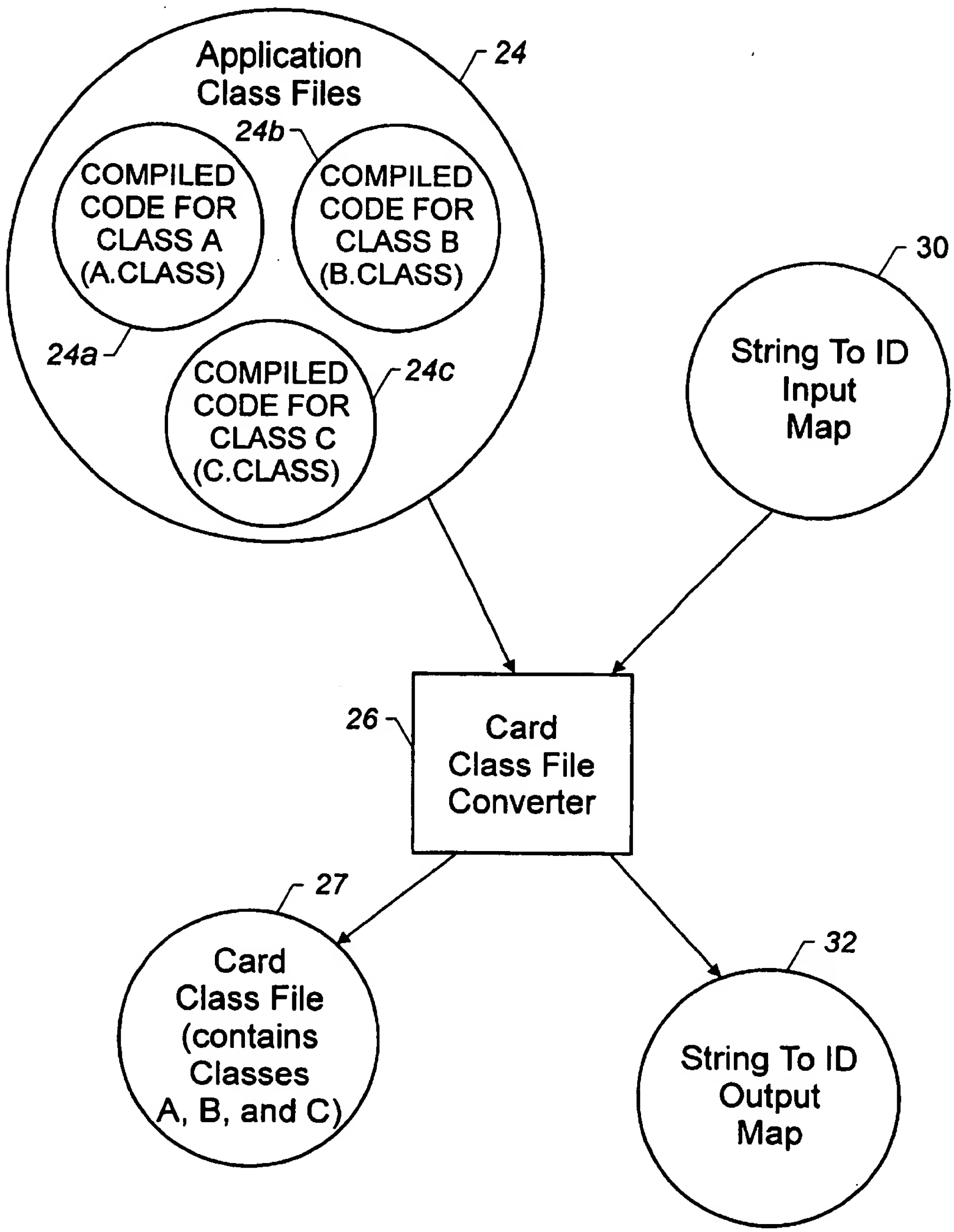

Application
Class Files
24
24b
COMPILED
CODE FOR
CLASS A
(A.CLASS)
COMPILED
CODE FOR
CLASS B
(B.CLASS)
24a
COMPILED
CODE FOR
CLASS C
(C.CLASS)
24c
30
String To ID
Input
Map
26
Card
Class File
Converter
27
Card
Class File
(contains
Classes
A, B, and C)
32
String To ID
Output
Map

FIGURE 3

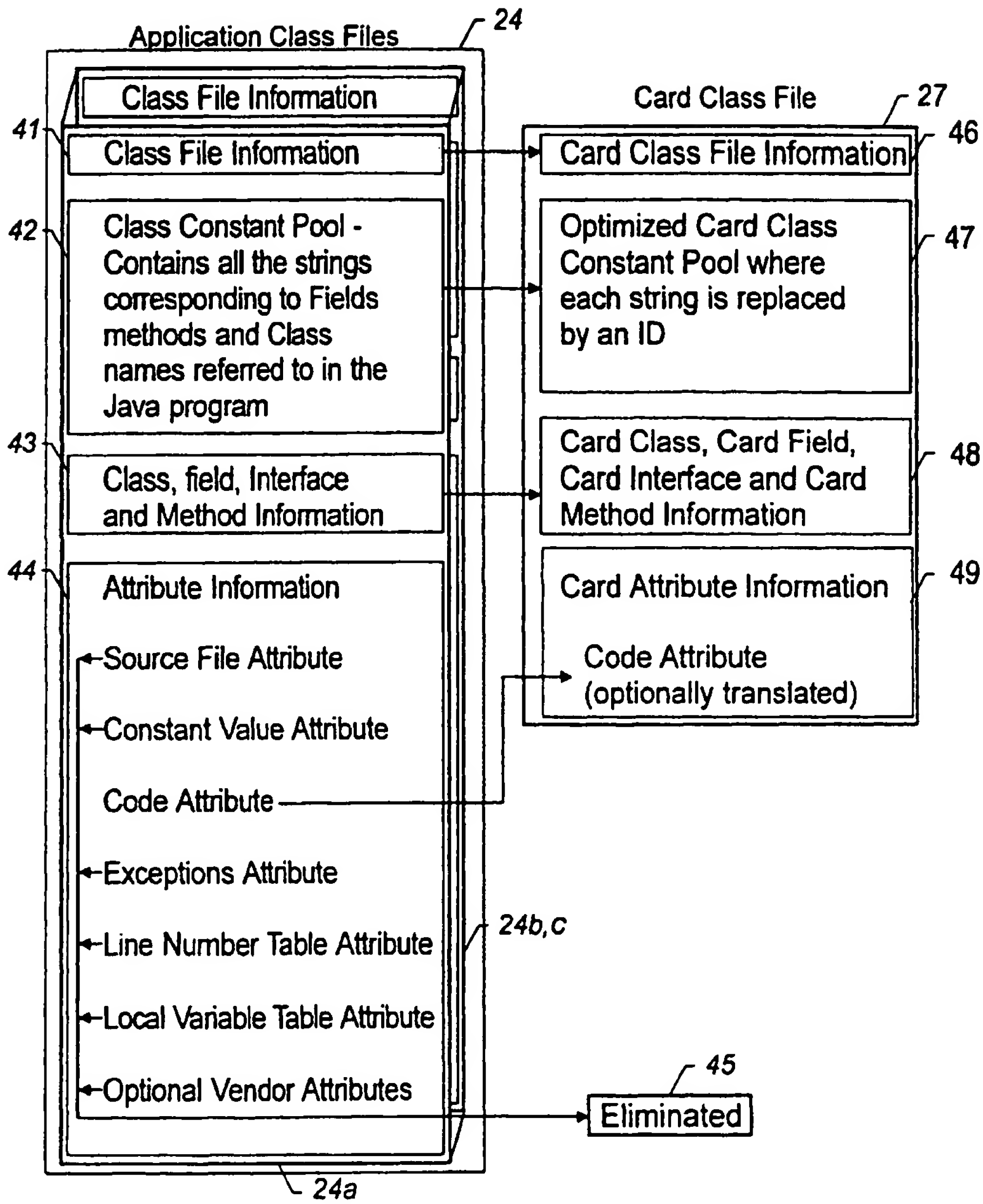

Application Class Files
24
Class File Information
41
Class File Information
42
Class Constant Pool - Contains all the strings corresponding to Fields methods and Class names referred to in the Java program
43
Class, field, Interface and Method Information
44
Attribute Information
Source File Attribute
Constant Value Attribute
Code Attribute
Exceptions Attribute
Line Number Table Attribute
Local Variable Table Attribute
Optional Vendor Attributes
24b,c
24a
Card Class File
27
46
Card Class File Information
Optimized Card Class Constant Pool where each string is replaced by an ID
47
Card Class, Card Field, Card Interface and Card Method Information
48
Card Attribute Information
49
Code Attribute (optionally translated)
45
Eliminated

FIGURE 4

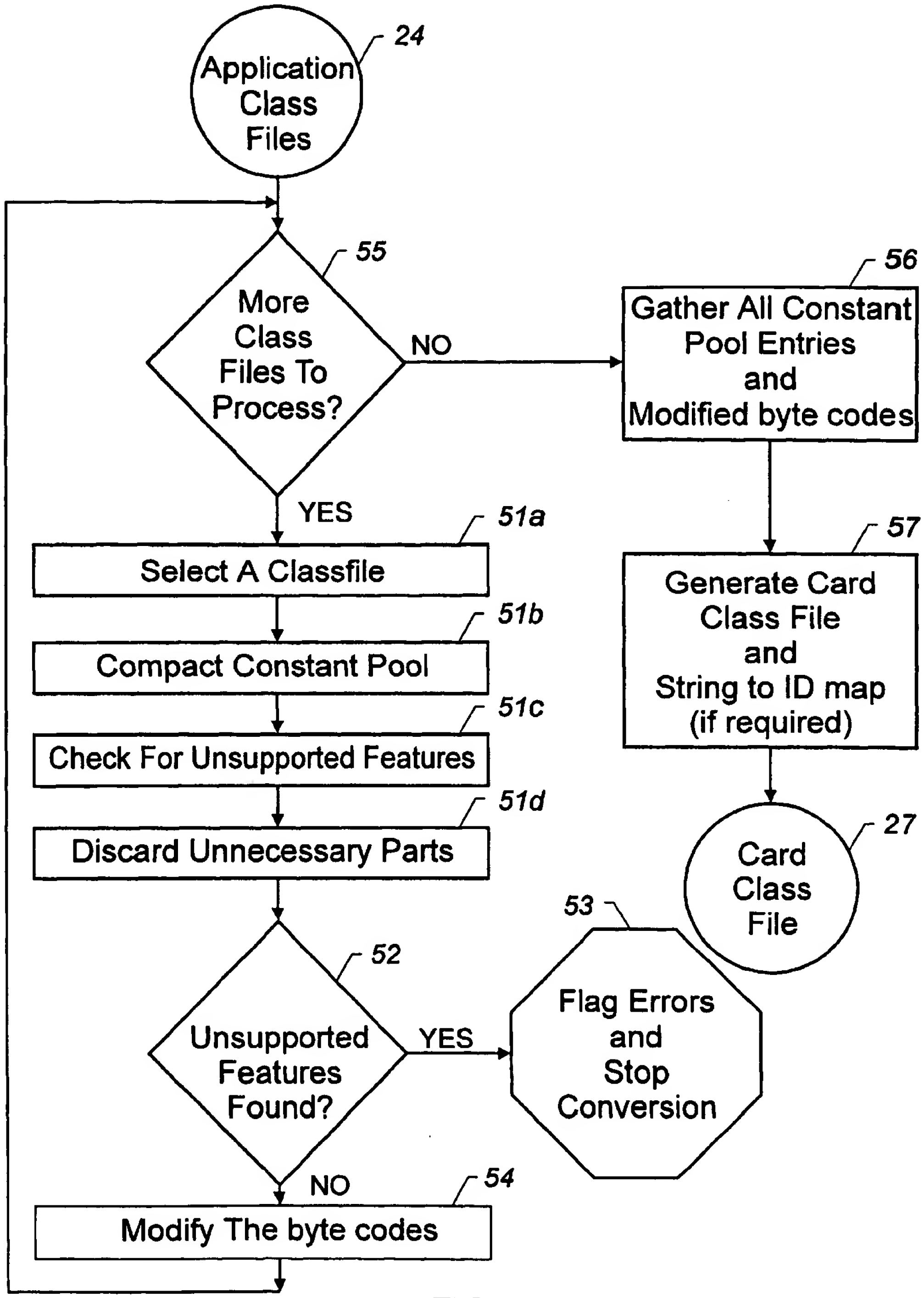

24
Application Class Files
55
More Class Files To Process?
NO
56
Gather All Constant Pool Entries and Modified byte codes
YES
51a
Select A Classfile
51b
Compact Constant Pool
51c
Check For Unsupported Features
51d
Discard Unnecessary Parts
57
Generate Card Class File and String to ID map (if required)
27
Card Class File
53
Flag Errors and Stop Conversion
52
Unsupported Features Found?
YES
NO
54
Modify The byte codes

FIGURE 5

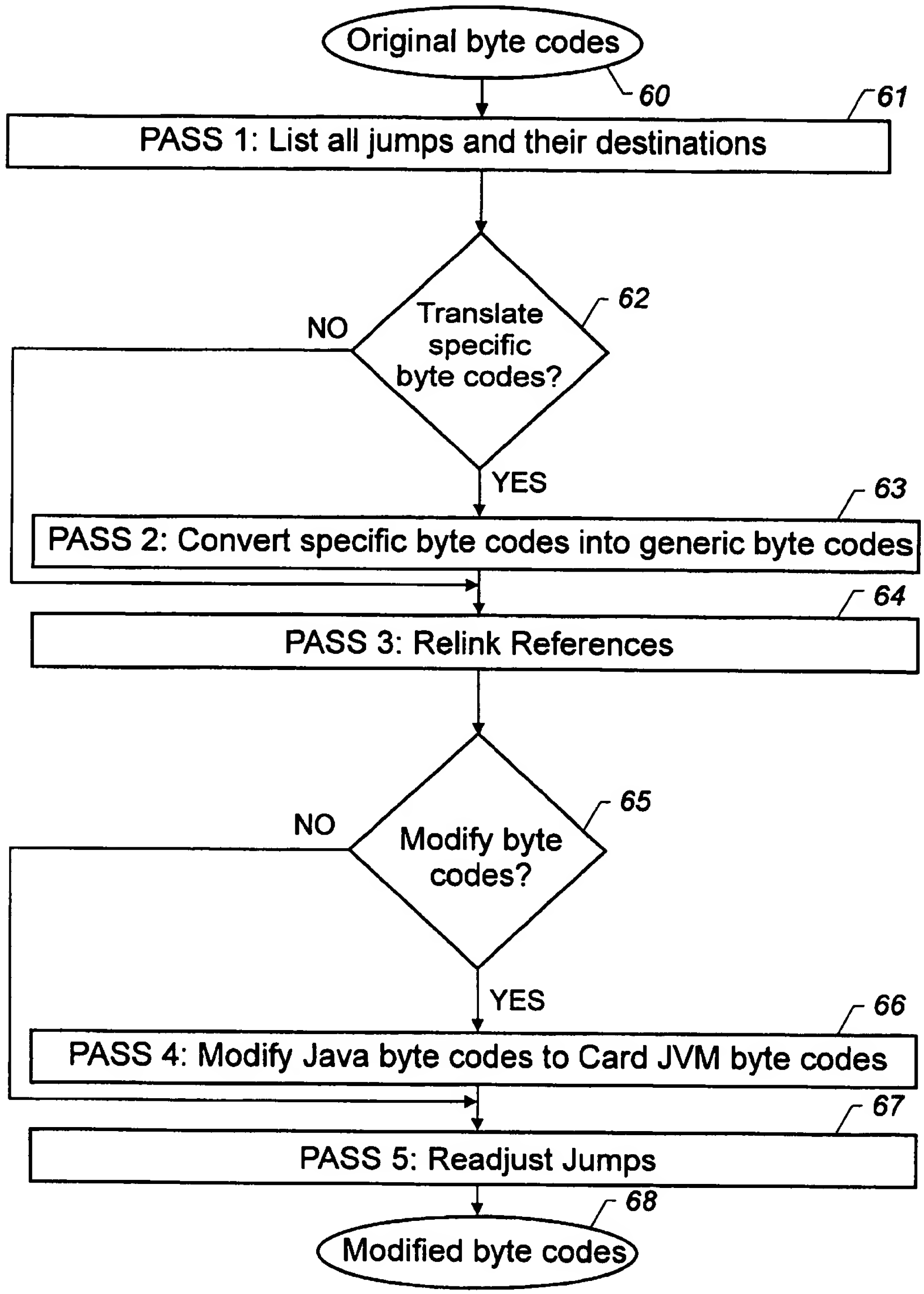

Original byte codes
60
61
PASS 1: List all jumps and their destinations
62
NO
Translate specific byte codes?
YES
63
PASS 2: Convert specific byte codes into generic byte codes
64
PASS 3: Relink References
65
NO
Modify byte codes?
YES
66
PASS 4: Modify Java byte codes to Card JVM byte codes
67
PASS 5: Readjust Jumps
68
Modified byte codes

FIGURE 6

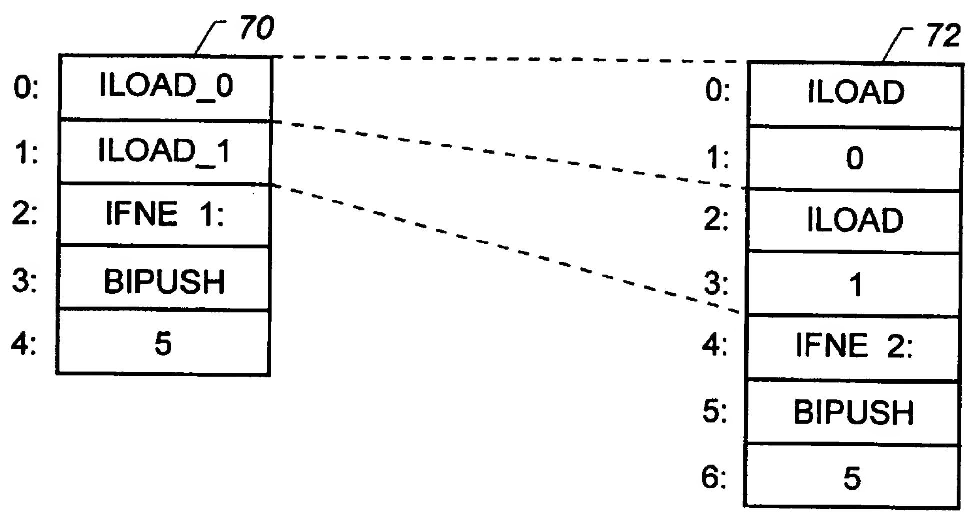

70
0:
ILOAD_0
1:
ILOAD_1
2:
IFNE 1:
3:
BIPUSH
4:
5
72
0:
ILOAD
1:
0
2:
ILOAD
3:
1
4:
IFNE 2:
5:
BIPUSH
6:
5

FIGURE 7

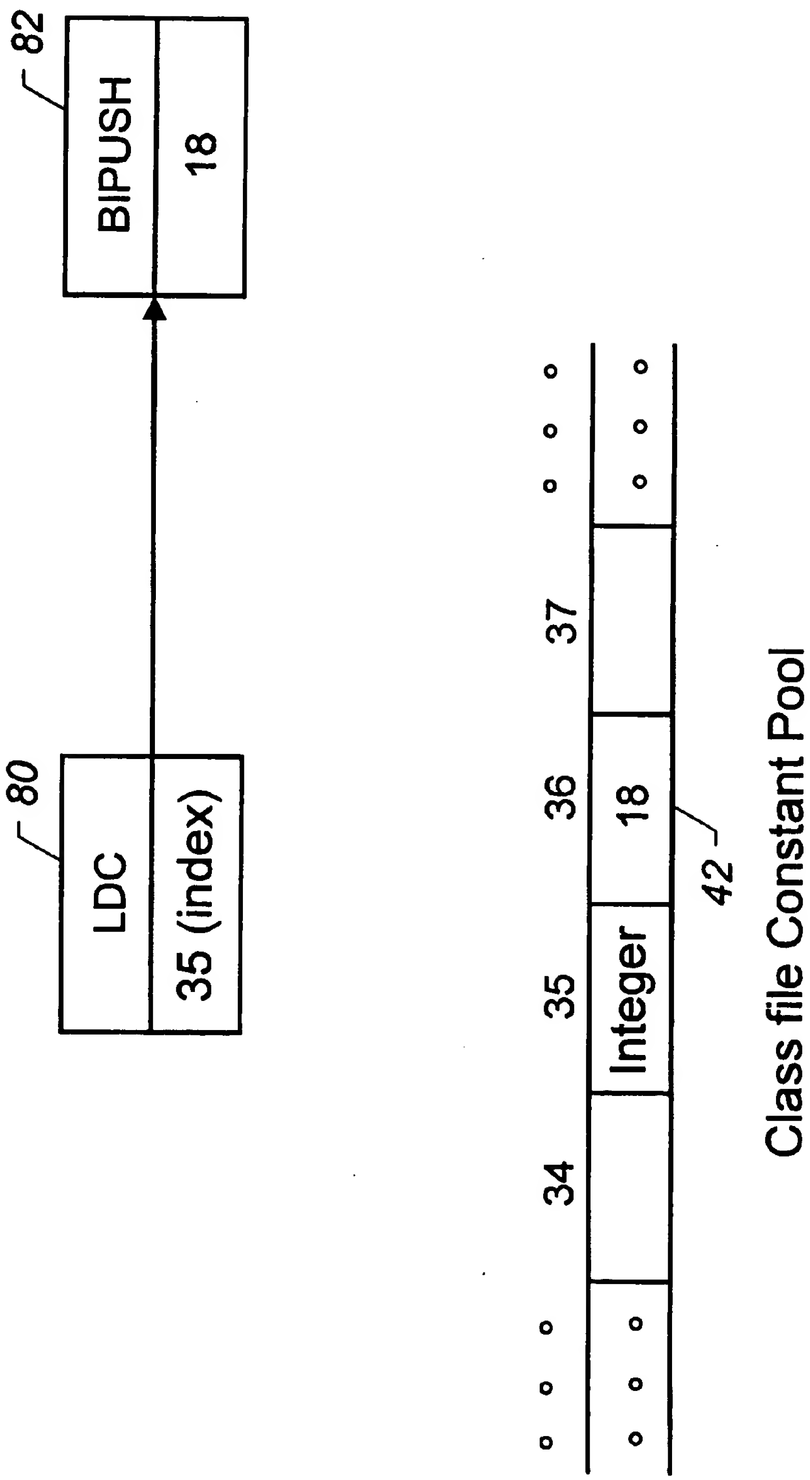

82
BIPUSH
18
80
LDC
35 (index)
34
35
36
37
Integer
18
42
Class file Constant Pool

FIGURE 8

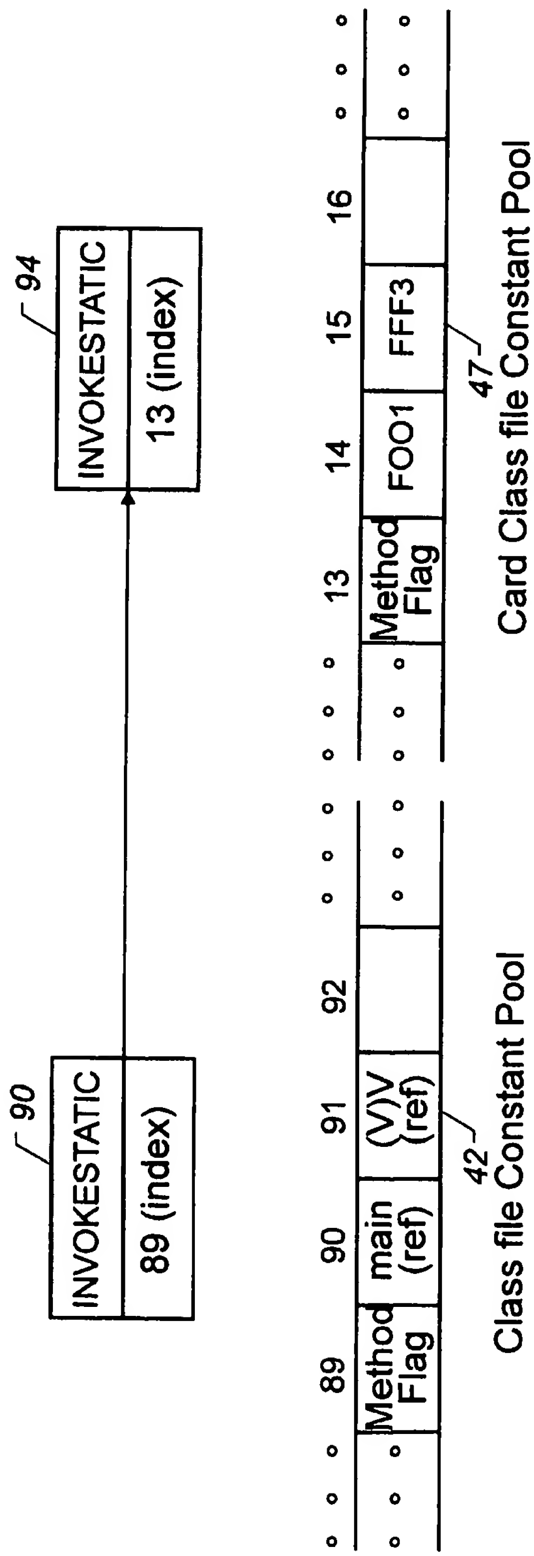

94
INVOKESTATIC
13 (index)
90
INVOKESTATIC
89 (index)
13
14
15
16
Method Flag
FOO1
FFF3
47
Card Class file Constant Pool
89
90
91
92
Method Flag
main (ref)
(V)V (ref)
42
Class file Constant Pool

FIGURE 9

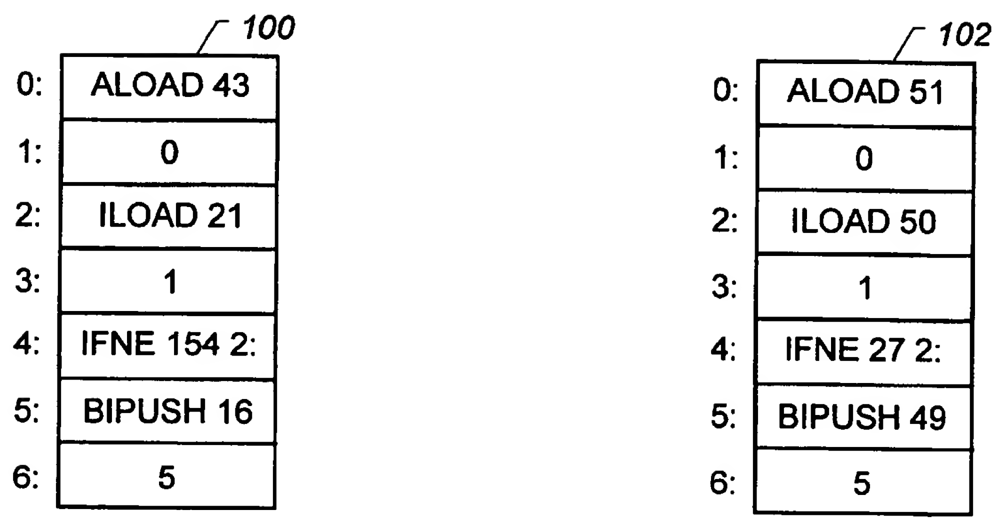

100
0: ALOAD 43
1: 0
2: ILOAD 21
3: 1
4: IFNE 154 2:
5: BIPUSH 16
6: 5
102
0: ALOAD 51
1: 0
2: ILOAD 50
3: 1
4: IFNE 27 2:
5: BIPUSH 49
6: 5

FIGURE 10

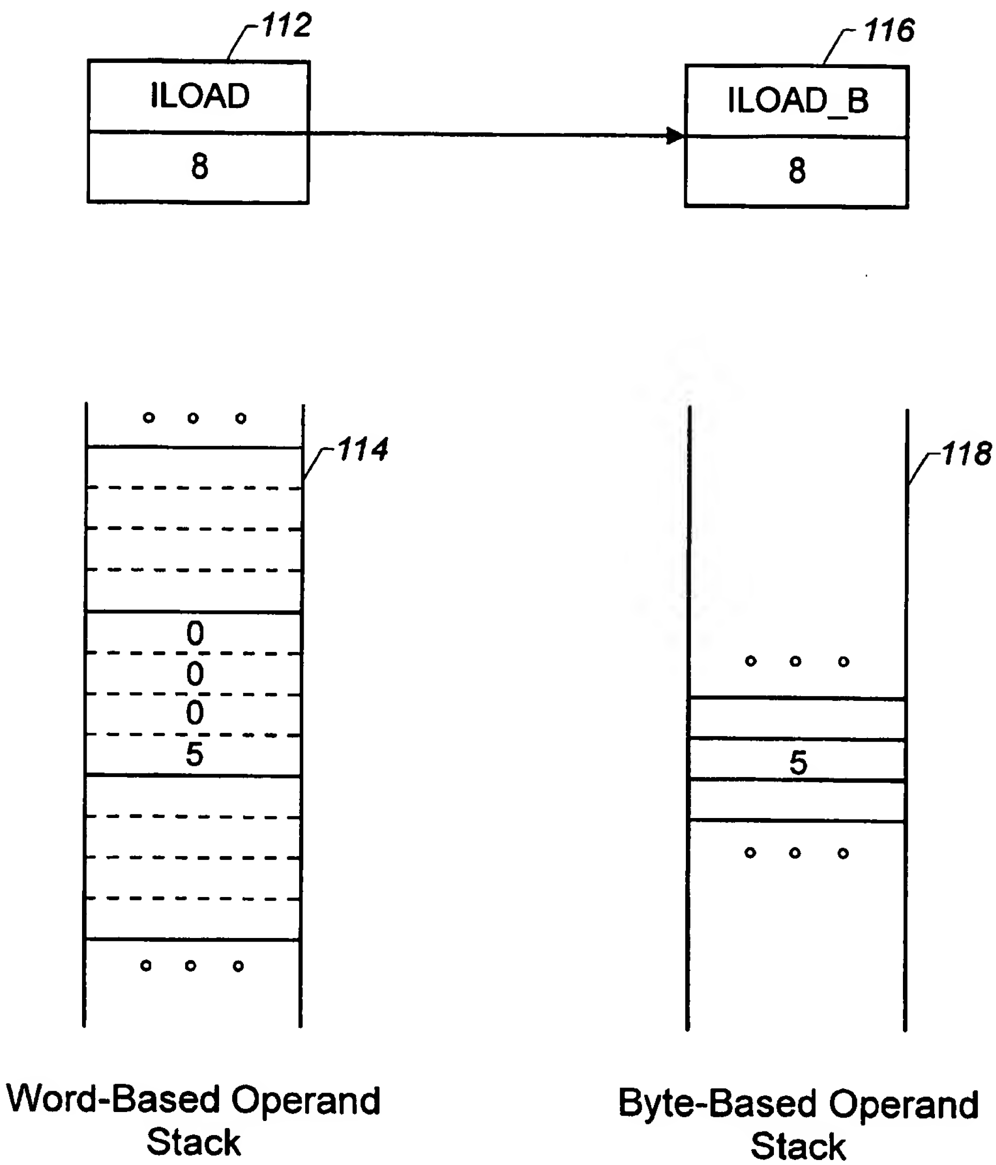

112
ILOAD
8
116
ILOAD_B
8
114
0
0
0
5
118
5
Word-Based Operand Stack
Byte-Based Operand Stack

FIGURE 11

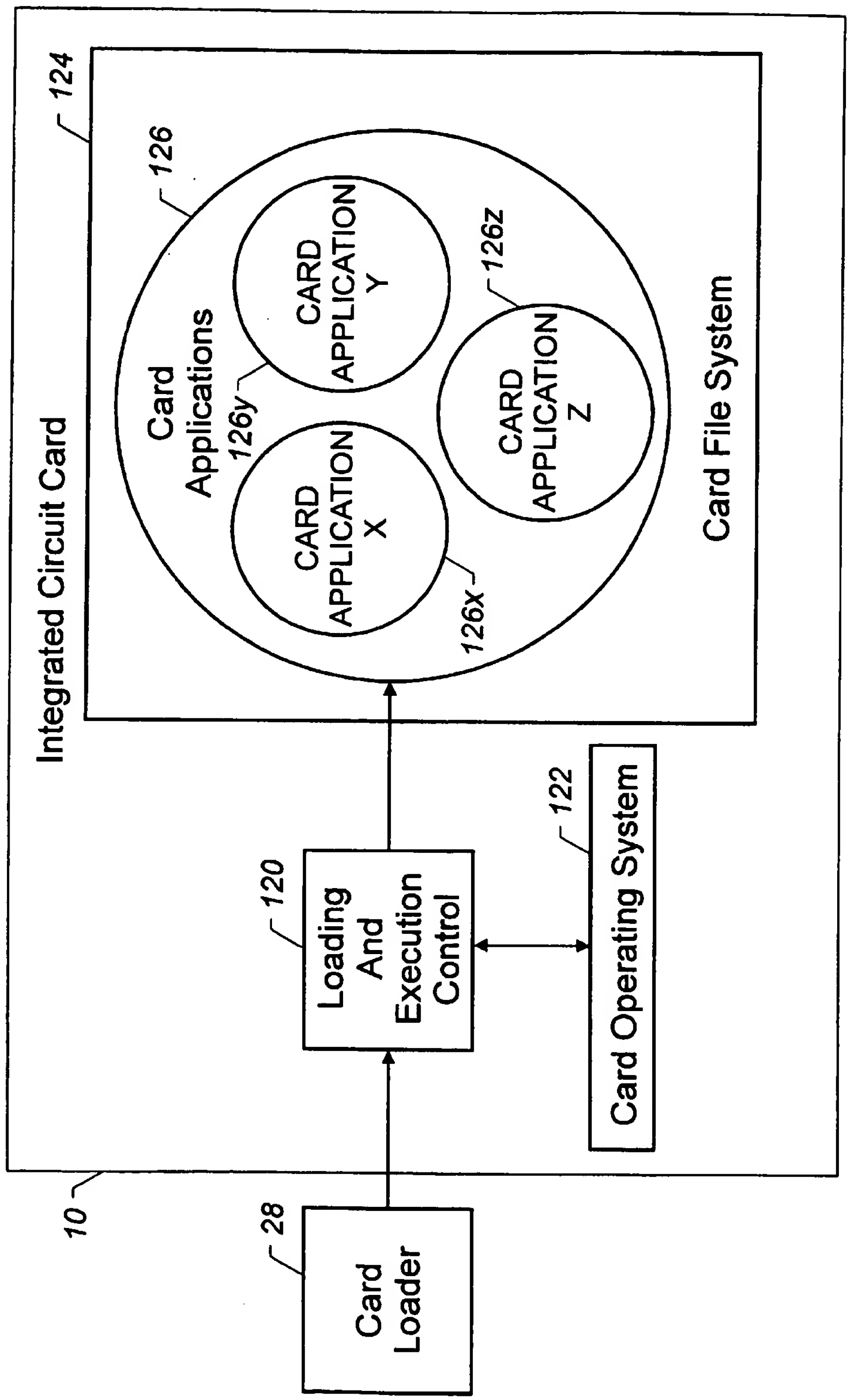

Integrated Circuit Card
124
126
Card Applications
CARD APPLICATION Y
126y
CARD APPLICATION X
126x
126z
CARD APPLICATION Z
Card File System
120
Loading And Execution Control
122
Card Operating System
10
28
Card Loader

FIGURE 12

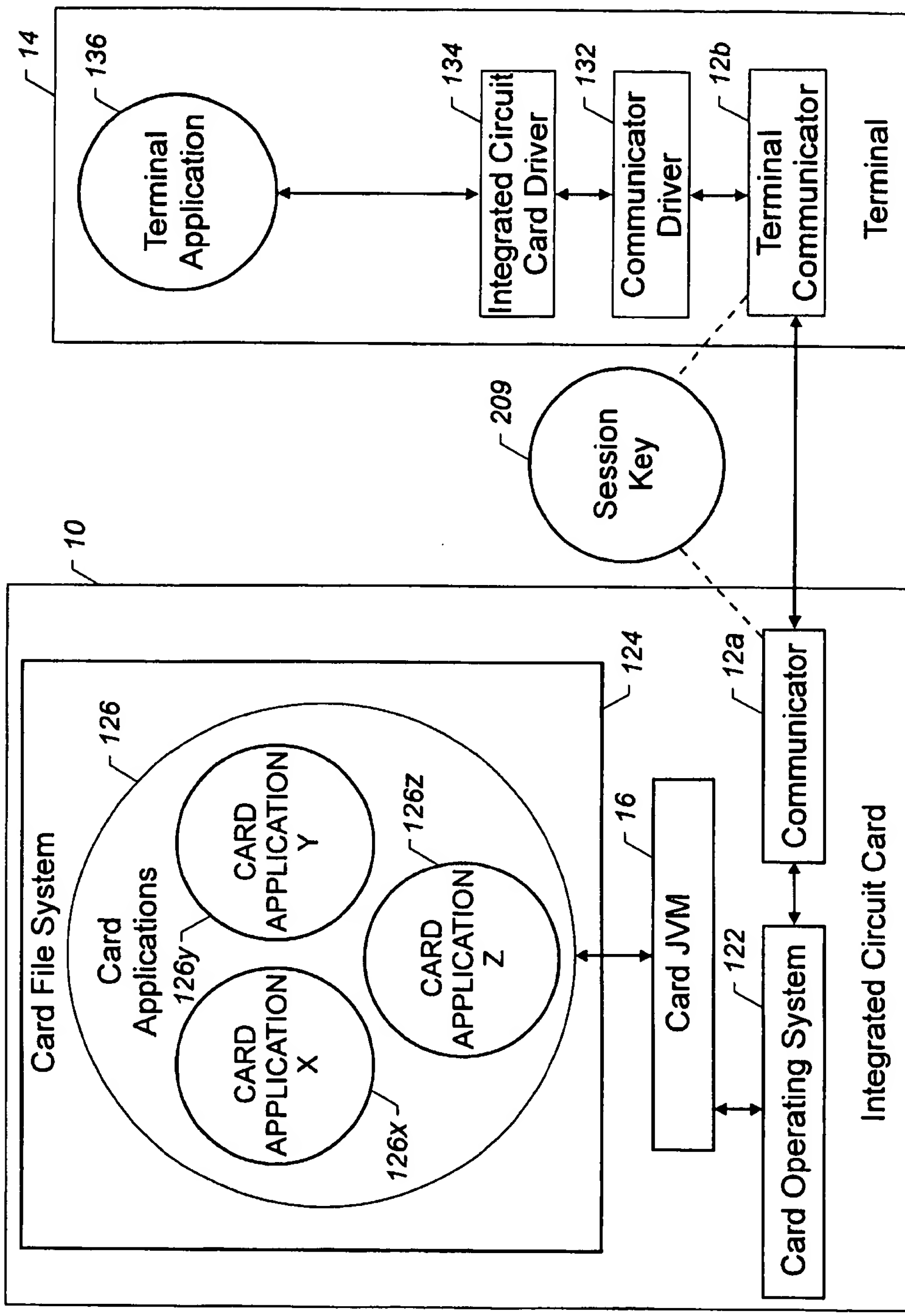

14
136
Terminal Application
134
Integrated Circuit Card Driver
132
Communicator Driver
12b
Terminal Communicator
Terminal
209
Session Key
10
Card File System
124
126
Card Applications
CARD APPLICATION Y
126y
CARD APPLICATION X
126x
126z
CARD APPLICATION Z
16
Card JVM
12a
Communicator
122
Card Operating System
Integrated Circuit Card

FIGURE 13

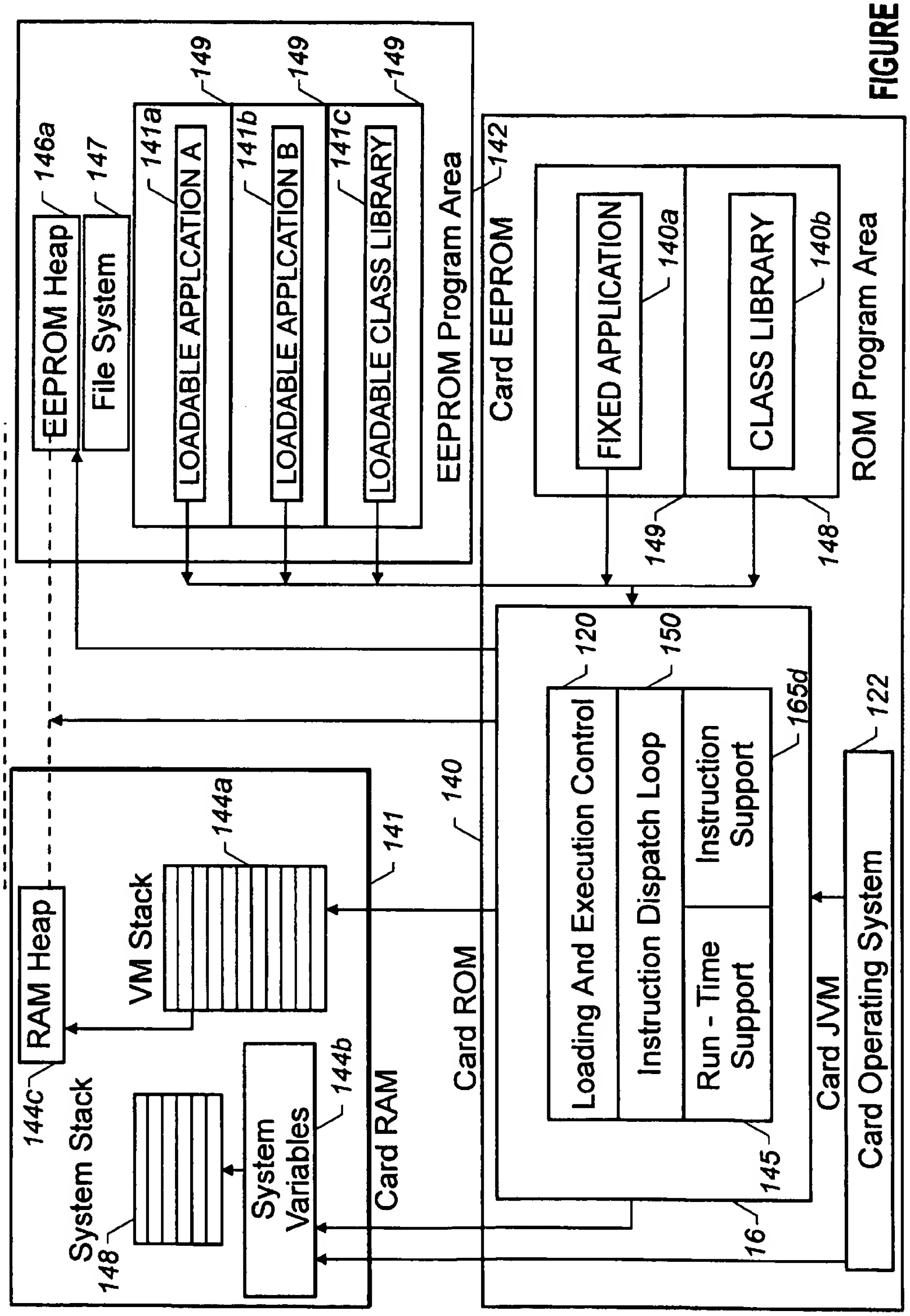
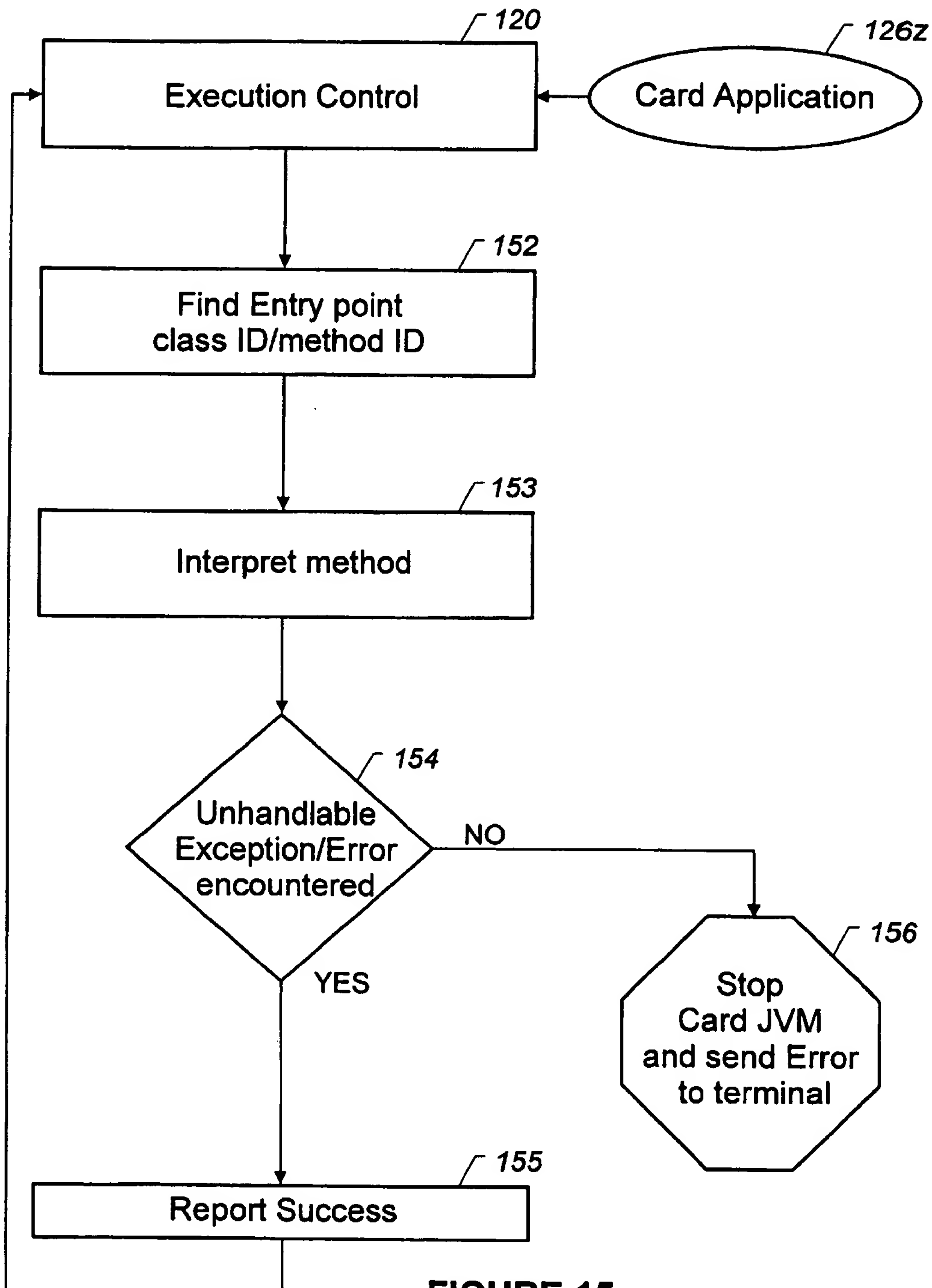

EEPROM Heap
146a
File System
147
LOADABLE APPLCATION A
141a
LOADABLE APPLCATION B
141b
LOADABLE CLASS LIBRARY
141c
149
EEPROM Program Area
Card EEPROM
142
FIXED APPLICATION
140a
CLASS LIBRARY
140b
ROM Program Area
148
Loading And Execution Control
120
Instruction Dispatch Loop
150
Instruction Support
165d
Run - Time Support
145
Card JVM
16
Card Operating System
122
Card ROM
140
RAM Heap
144c
VM Stack
144a
System Stack
148
System Variables
144b
Card RAM
141

FIGURE 14



120
126z
Execution Control
Card Application
152
Find Entry point
class ID/method ID
153
Interpret method
154
Unhandlable
Exception/Error
encountered
NO
YES
156
Stop
Card JVM
and send Error
to terminal
155
Report Success

FIGURE 15

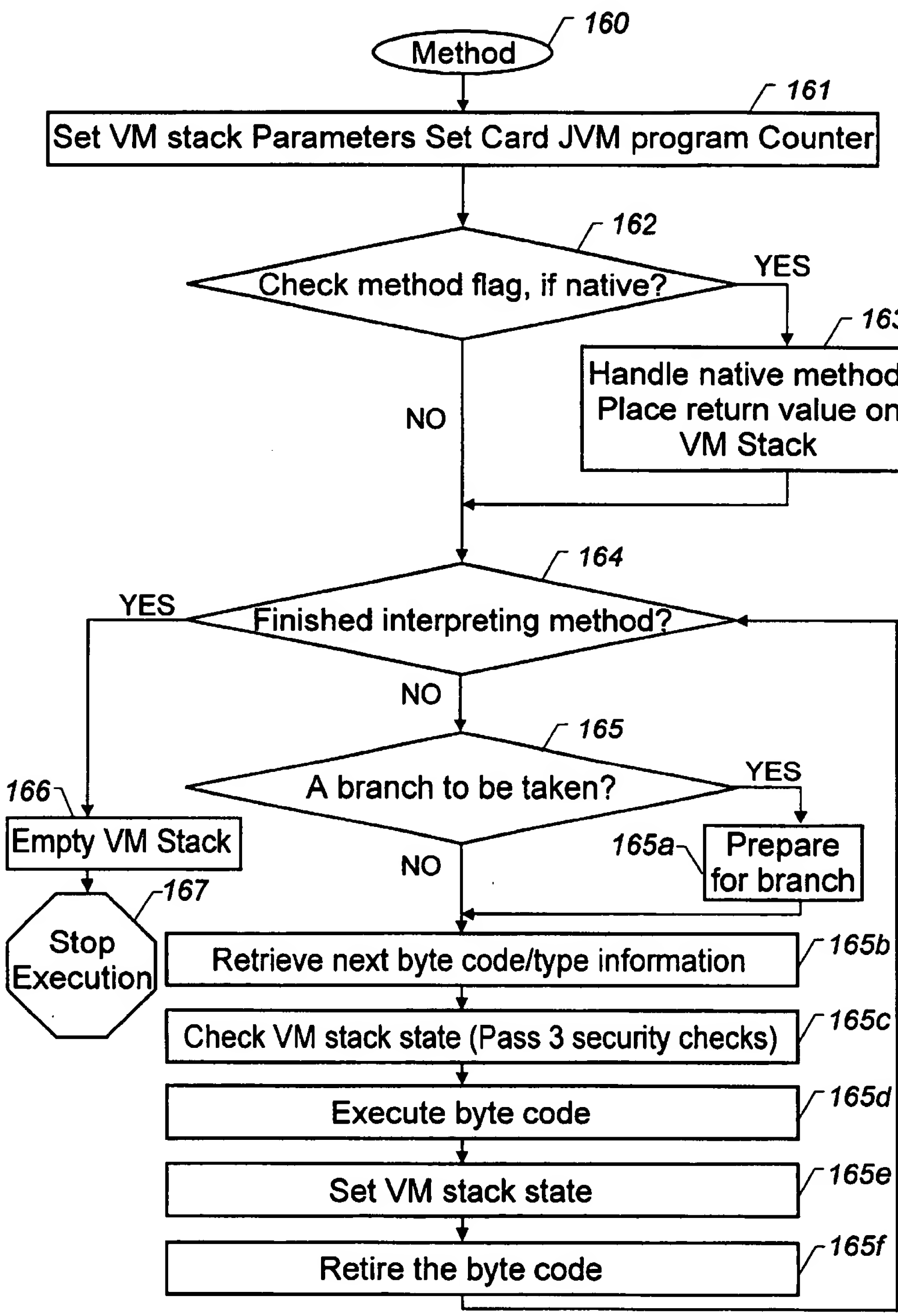

160
Method
161
Set VM stack Parameters Set Card JVM program Counter
162
Check method flag, if native?
YES
163
Handle native method Place return value on VM Stack
NO
164
Finished interpreting method?
YES
NO
165
A branch to be taken?
YES
166
Empty VM Stack
165a
Prepare for branch
NO
167
Stop Execution
Retrieve next byte code/type information
165b
Check VM stack state (Pass 3 security checks)
165c
Execute byte code
165d
Set VM stack state
165e
Retire the byte code
165f

FIGURE 16

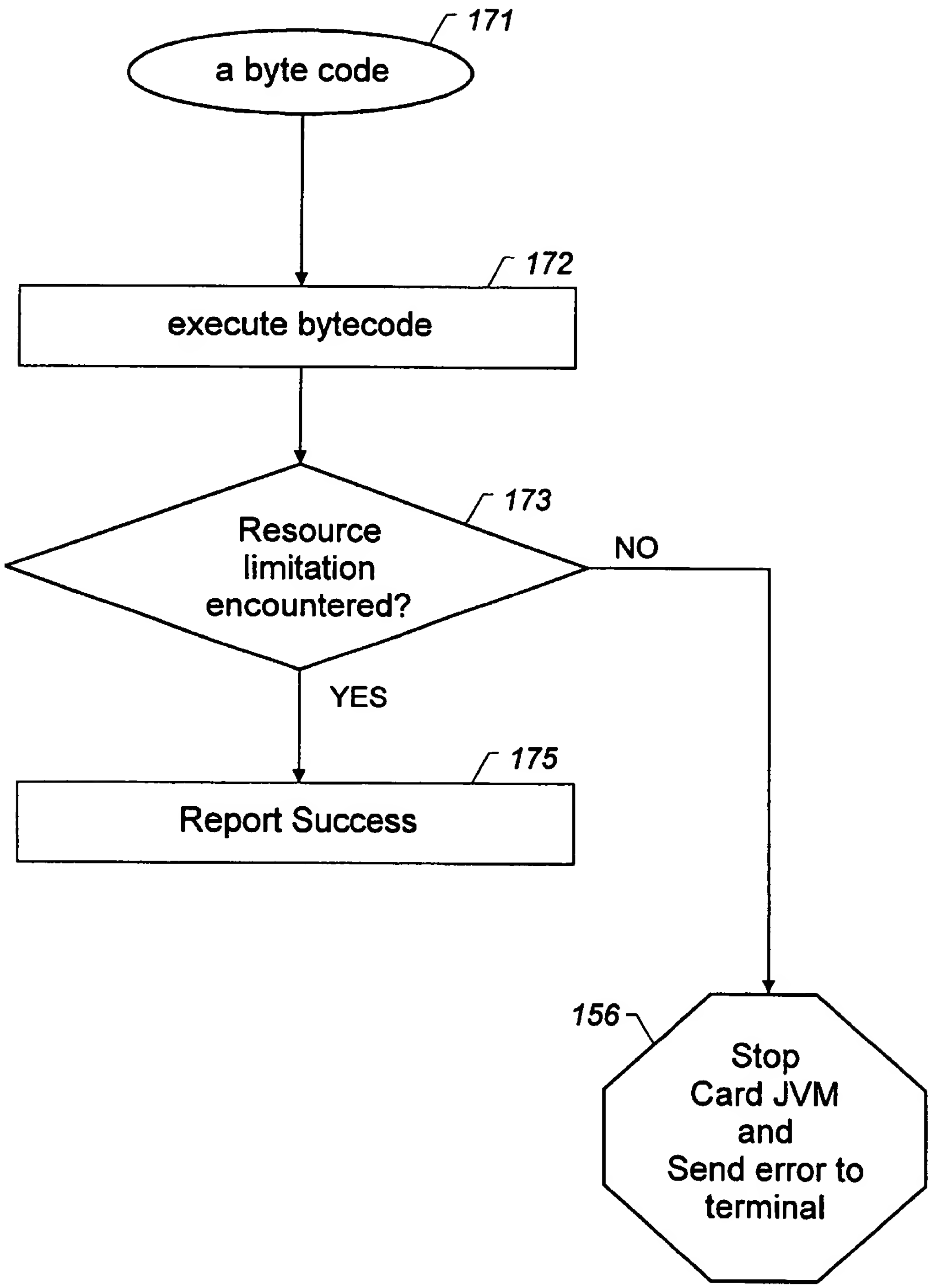

171
a byte code
172
execute bytecode
173
Resource limitation encountered?
NO
YES
175
Report Success
156
Stop
Card JVM
and
Send error to
terminal

FIGURE 17

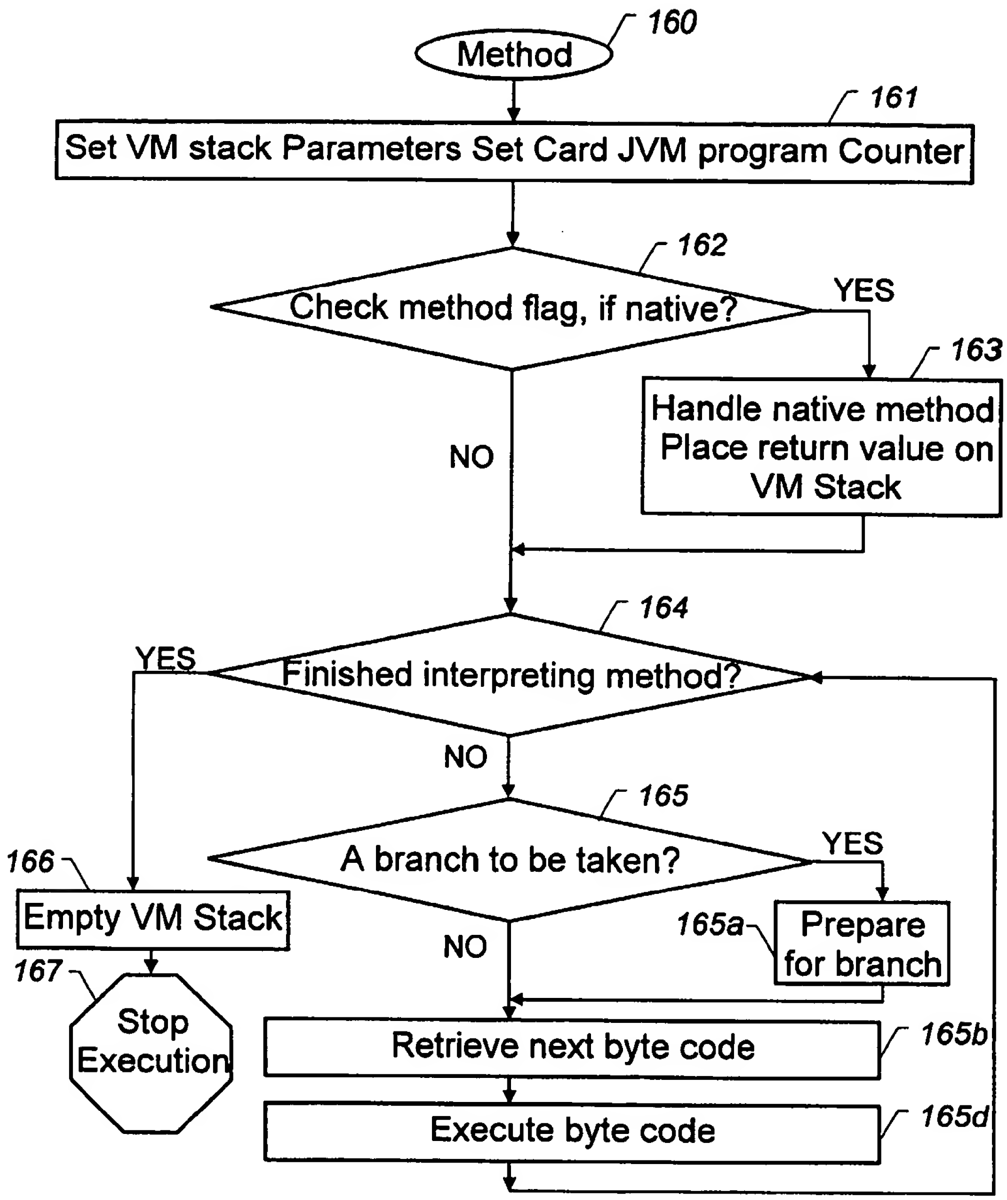
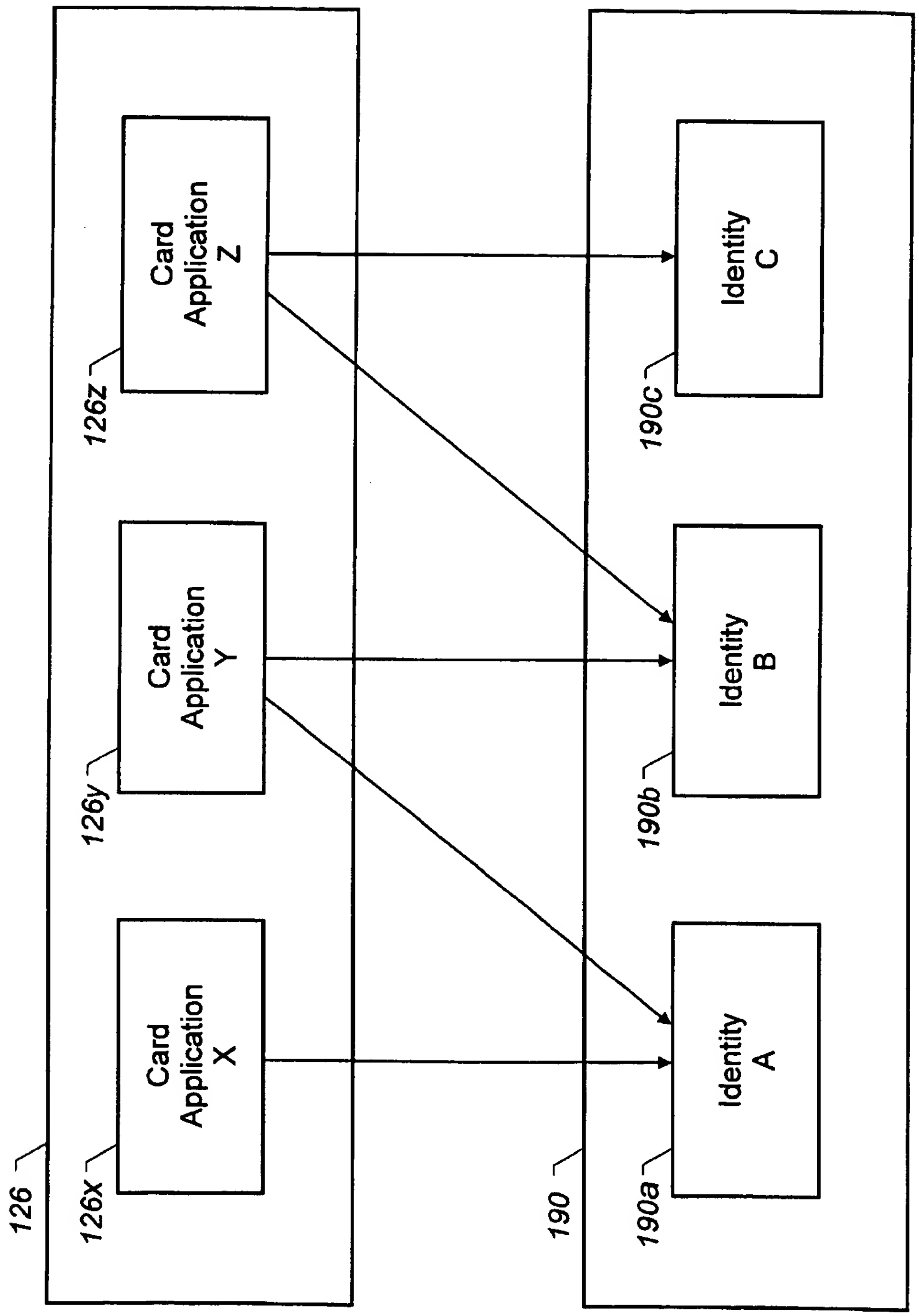

160
Method
161
Set VM stack Parameters Set Card JVM program Counter
162
Check method flag, if native?
YES
163
Handle native method Place return value on VM Stack
NO
164
Finished interpreting method?
YES
NO
165
A branch to be taken?
YES
166
Empty VM Stack
165a
Prepare for branch
NO
167
Stop Execution
Retrieve next byte code
165b
Execute byte code
165d

FIGURE 18



126
126x
Card Application X
126y
Card Application Y
126z
Card Application Z
190
190a
Identity A
190b
Identity B
190c
Identity C

FIGURE 19

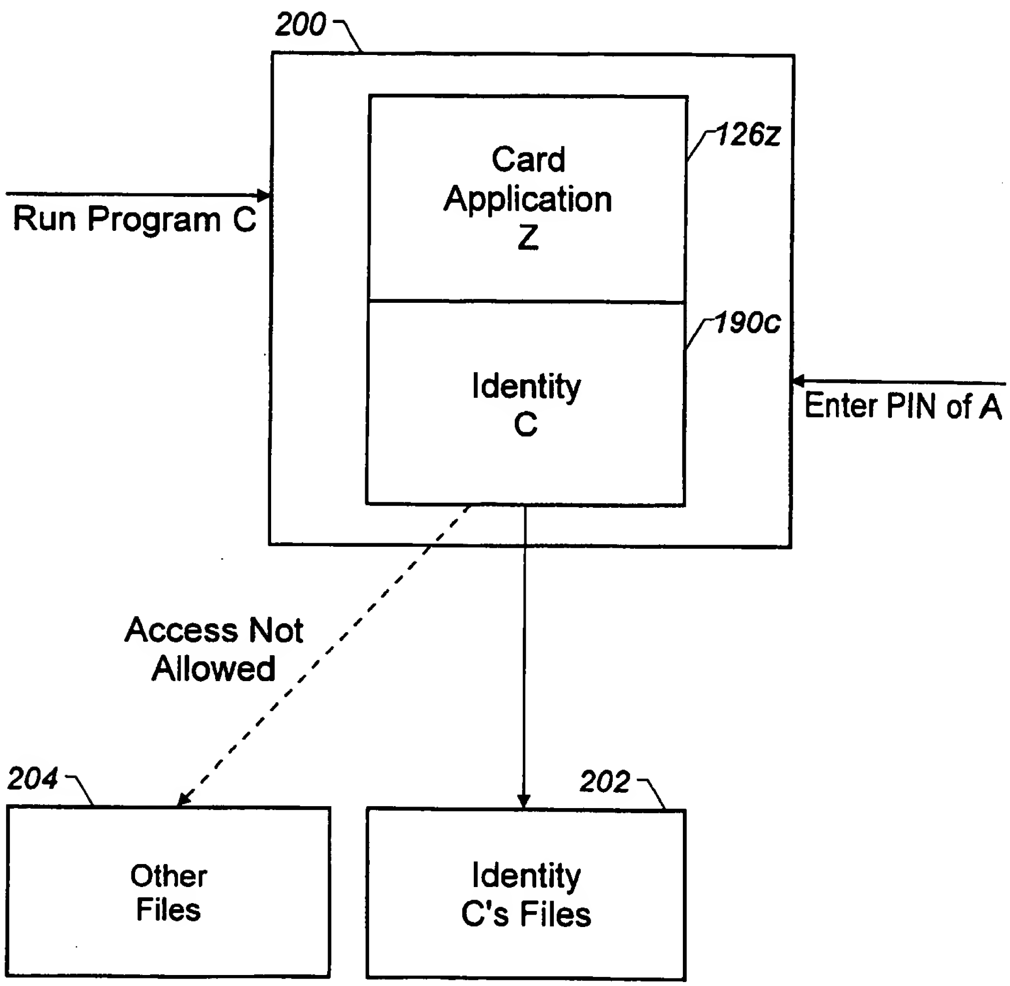

200
Card
Application
Z
126z
Run Program C
Identity
C
190c
Enter PIN of A
Access Not
Allowed
204
Other
Files
202
Identity
C's Files

FIGURE 20

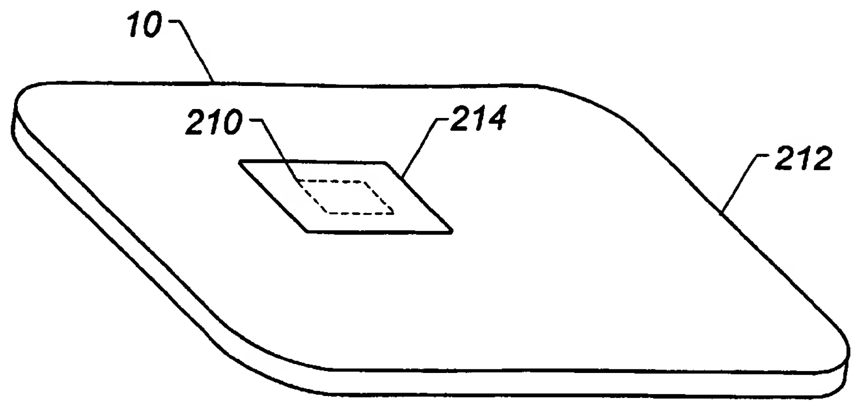


FIGURE 21

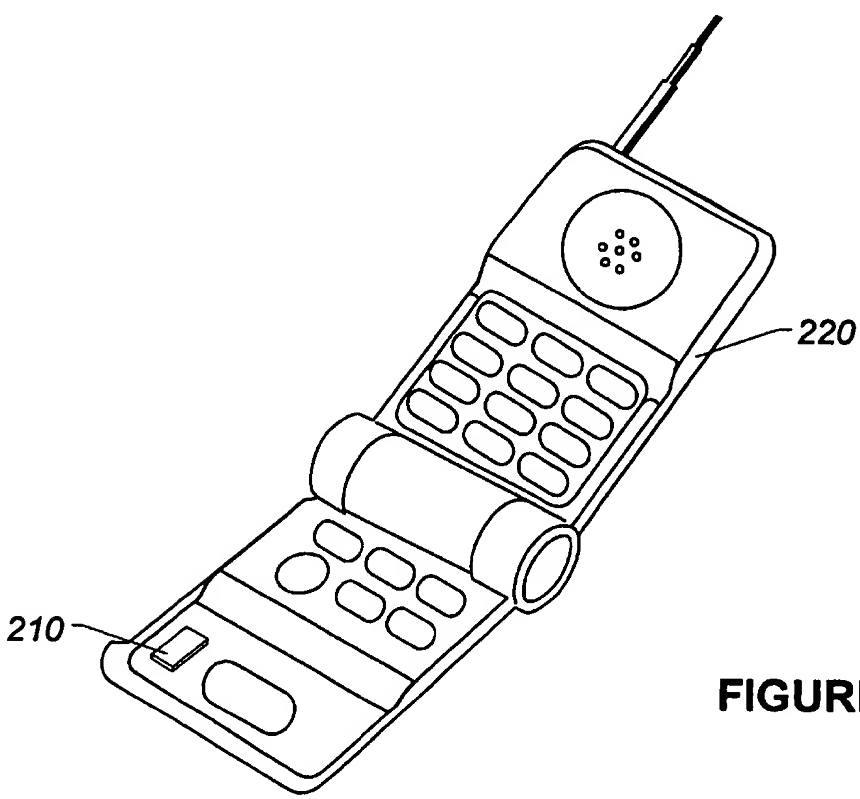


FIGURE 22

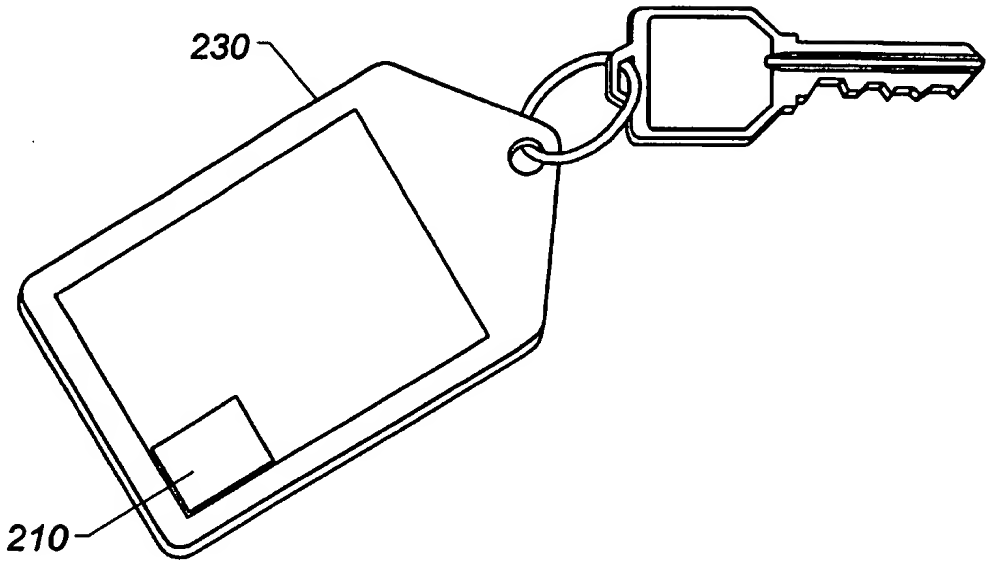


FIGURE 23

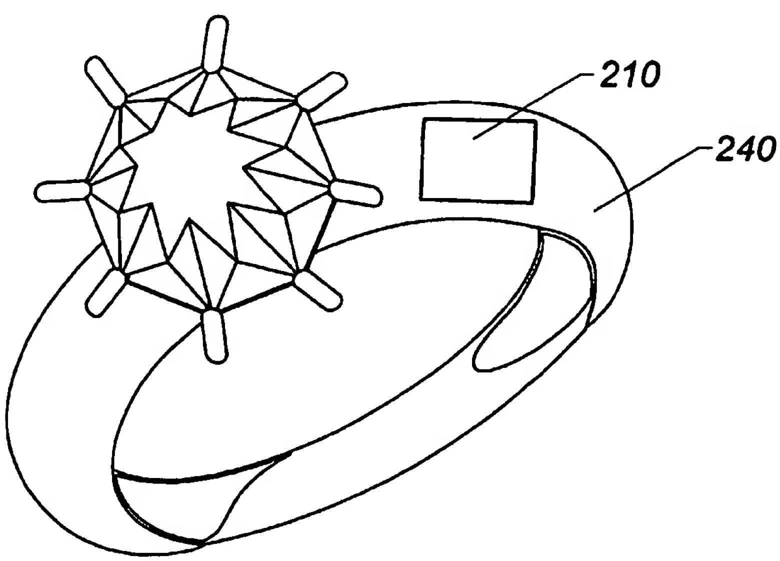


FIGURE 24

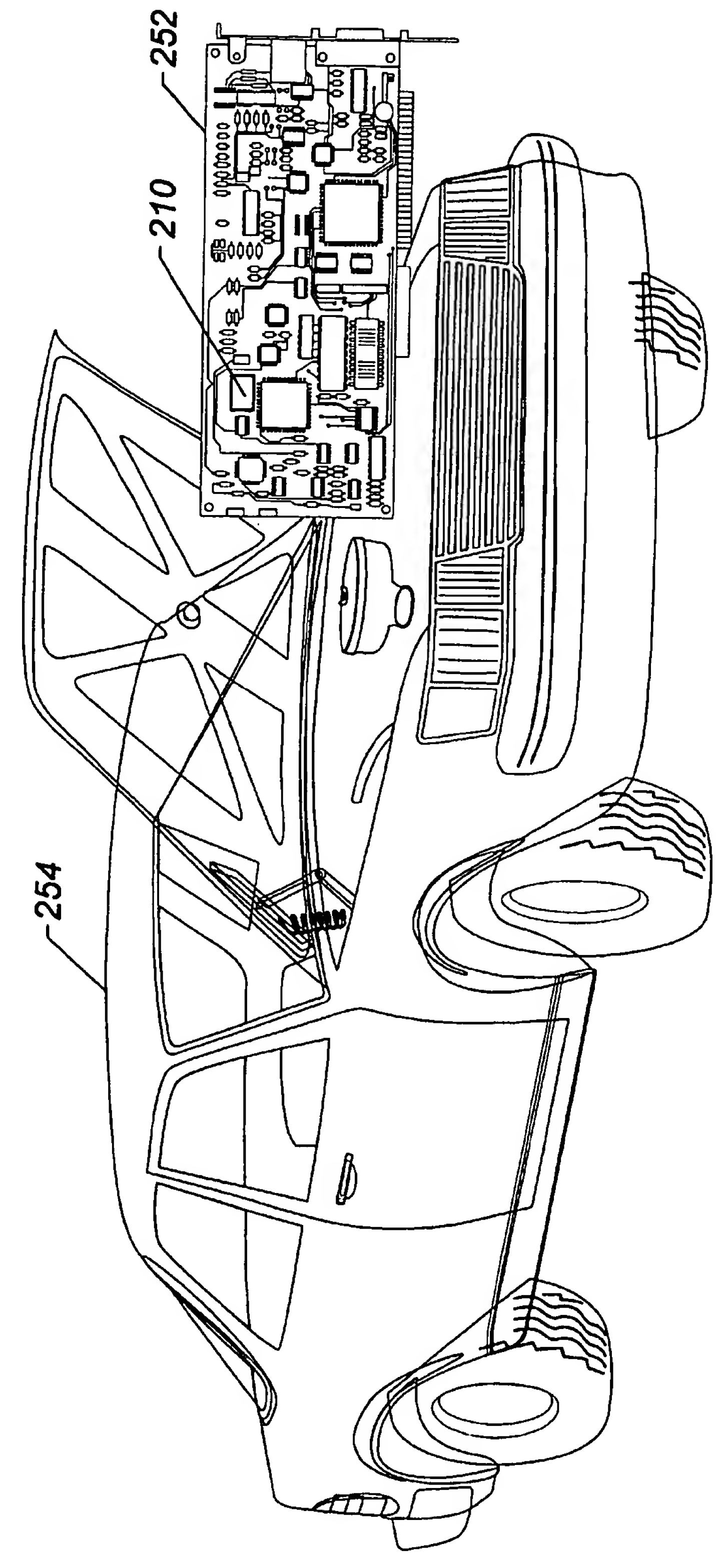


FIGURE 25

USING A HIGH LEVEL PROGRAMMING LANGUAGE WITH A MICROCONTROLLER

Under 35 U.S.C. § 119(e), this application claims benefit of prior U.S. provisional application Serial No. 60/029,057, filed Oct. 25, 1996.

BACKGROUND OF THE INVENTION

This invention relates in general to the field of programming, and more particularly to using a high level programming language with a smart card or a microcontroller.

Software applications written in the Java high-level programming language have been so designed that an application written in Java can be run on many different computer brands or computer platforms without change. This is accomplished by the following procedure. When a Java application is written, it is compiled into "Class" files containing byte codes that are instructions for a hypothetical computer called a Java Virtual Machine. An implementation of this virtual machine is written for each platform that is supported. When a user wishes to run a particular Java application on a selected platform, the class files compiled from the desired application is loaded onto the selected platform. The Java virtual machine for the selected platform is run, and interprets the byte codes in the class file, thus effectively running the Java application.

Java is described in the following references which are hereby incorporated by reference: (1) Arnold, Ken, and James Gosling, "The Java Programming Language," Addison-Wesley, 1996; (2) James Gosling, Bill Joy, and Guy Steele, "The Java Language Specification," Sun Microsystems, 1996, (web site: http://java.sun.com/doc/language_specification); (3) James Gosling and Henry McGilton, "The Java Language Environment: A White Paper," Sun Microsystems, 1995 (web site: http://java.sun.com/doc/language_environment/); and (4) Tim Lindholm and Frank Yellin, "The Java Virtual Machine Specification," Addison-Wesley, 1997. These texts among many others describe how to program using Java.

In order for a Java application to run on a specific platform, a Java virtual machine implementation must be written that will run within the constraints of the platform, and a mechanism must be provided for loading the desired Java application on the platform, again keeping within the constraints of this platform.

Conventional platforms that support Java are typically microprocessor-based computers, with access to relatively large amounts of memory and hard disk storage space. Such microprocessor implementations frequently are used in desktop and personal computers. However, there are no conventional Java implementations on microcontrollers, as would typically be used in a smart card.

Microcontrollers differ from microprocessors in many ways. For example, a microprocessor typically has a central processing unit that requires certain external components (e.g., memory, input controls and output controls) to function properly. A typical microprocessor can access from a megabyte to a gigabyte of memory, and is capable of

processing 16, 32, or 64 bits of information or more with a single instruction. In contrast to the microprocessor, a microcontroller includes a central processing unit, memory and other functional elements, all on a single semiconductor substrate, or integrated circuit (e.g., a "chip"). As compared to the relatively large external memory accessed by the microprocessor, the typical microcontroller accesses a much smaller memory. A typical microcontroller can access one to sixty-four kilobytes of built-in memory, with sixteen kilobytes being very common.

There are generally three different types of memory used: random access memory (RAM), read only memory (ROM), and electrically erasable programmable read only memory (EEPROM). In a microcontroller, the amount of each kind of memory available is constrained by the amount of space on the integrated circuit used for each kind of memory. Typically, RAM takes the most space, and is in shortest supply. ROM takes the least space, and is abundant. EEPROM is more abundant than RAM, but less than ROM.

Each kind of memory is suitable for different purposes. Although ROM is the least expensive, it is suitable only for data that is unchanging, such as operating system code. EEPROM is useful for storing data that must be retained when power is removed, but is extremely slow to write. RAM can be written and read at high speed, but is expensive and data in RAM is lost when power is removed. A microprocessor system typically has relatively little ROM and EEPROM, and has 1 to 128 megabytes of RAM, since it is not constrained by what will fit on a single integrated circuit device, and often has access to an external disk memory system that serves as a large writable, non-volatile storage area at a lower cost than EEPROM. However, a microcontroller typically has a small RAM of 0.1 to 2.0 K, 2K to 8K of EEPROM, and 8K–56K of ROM.

Due to the small number of external components required and their small size, microcontrollers frequently are used in integrated circuit cards, such as smart cards. Such smart cards come in a variety of forms, including contact-based cards, which must be inserted into a reader to be used, and contactless cards, which need not be inserted. In fact, microcontrollers with contactless communication are often embedded into specialized forms, such as watches and rings, effectively integrating the functionality of a smart card in an ergonomically attractive manner.

Because of the constrained environment, applications for smart cards are typically written in a low level programming language (e.g., assembly language) to conserve memory.

The integrated circuit card is a secure, robust, tamper-resistant and portable device for storing data. The integrated circuit card is the most personal of personal computers because of its small size and because of the hardware and software data security features unique to the integrated circuit card.

The primary task of the integrated circuit card and the microcontroller on the card is to protect the data stored on the card. Consequently, since its invention in 1974, integrated circuit card technology has been closely guarded on these same security grounds. The cards were first used by French banks as debit cards. In this application, before a financial transaction based on the card is authorized, the card user must demonstrate knowledge of a 4-digit personal identification number (PIN) stored in the card in addition to being in possession of the card. Any information that might contribute to discovering the PIN number on a lost or stolen card was blocked from public distribution. In fact, since nobody could tell what information might be useful in this

regard, virtually all information about integrated circuit cards was withheld.

Due to the concern for security, applications written for integrated circuit cards have unique properties. For example, each application typically is identified with a particular owner or identity. Because applications typically are written in a low-level programming language, such as assembly language, the applications are written for a particular type of microcontroller. Due to the nature of low level programming languages, unauthorized applications may access data on the integrated circuit card. Programs written for an integrated circuit card are identified with a particular identity so that if two identities want to perform the same programming function there must be two copies of some portions of the application on the microcontroller of the integrated circuit card.

Integrated circuit card systems have historically been closed systems. An integrated circuit card contained a dedicated application that was handcrafted to work with a specific terminal application. Security checking when an integrated circuit card was used consisted primarily of making sure that the card application and the terminal application were a matched pair and that the data on the card was valid.

As the popularity of integrated circuit cards grew, it became clear that integrated circuit card users would be averse to carrying a different integrated circuit card for each integrated circuit card application. Therefore, multiple cooperating applications began to be provided on single provider integrated circuit cards. Thus, for example, an automated teller machine (ATM) access card and a debit card may coexist on a single integrated circuit card platform. Nevertheless, this was still a closed system since all the applications in the terminal and the card were built by one provider having explicit knowledge of the other providers.

The paucity of information about integrated circuit cards—particularly information about how to communicate with them and how to program them—has impeded the general application of the integrated circuit card. However, the advent of public digital networking (e.g., the Internet and the World Wide Web) has opened new domains of application for integrated circuit cards. In particular, this has lead to a need to load new applications on the card that do not have explicit knowledge of the other providers, but without the possibility of compromising the security of the card. However, typically, this is not practical with conventional cards that are programmed using low level languages.

SUMMARY OF THE INVENTION

In general, in one aspect, the invention features an integrated circuit card for use with a terminal. The integrated circuit card includes a memory that stores an interpreter and an application that has a high level programming language format. A processor of the card is configured to use the interpreter to interpret the application for execution and to use a communicator of the card to communicate with the terminal.

Among the advantages of the invention are one or more of the following. New applications may be downloaded to a smart card without compromising the security of the smart card. These applications may be provided by different companies loaded at different times using different terminals. Security is not compromised since the applications are protected against unauthorized access of any application code or data by the security features provided by the Java virtual machine. Smart card applications can be created in high level languages such as Java and Eiffel, using powerful mainstream program development tools. New applications can be quickly prototyped and downloaded to a smart card in a matter of hours without resorting to soft masks. Embedded systems using microcontrollers can also gain many of these advantages for downloading new applications, high level program development, and rapid prototyping by making use of this invention.

Implementations of the invention may include one or more of the following. The high level programming language format of the application may have a class file format and may have a Java programming language format. The processor may be a microcontroller. At least a portion of the memory may be located in the processor.

The application may have been processed from a second application that has a string of characters, and the string of characters may be represented in the first application by an identifier (e.g., an integer).

The processor may be also configured to receive a request from a requester (e.g., a processor or a terminal) to access an element (e.g., an application stored in the memory, data stored in the memory or the communicator) of the card, after receipt of the request, interact with the requester to authenticate an identity of the requester, and based on the identity, selectively grant access to the element.

The memory may also store an access control list for the element. The access control list furnishes an indication of types of access to be granted to the identity, and based on the access control list, the processor selectively grants specific types of access (e.g., reading data, writing data, appending data, creating data, deleting data or executing an application) to the requester.

The application may be one of a several applications stored in the memory. The processor may be further configured to receive a request from a requester to access one of the plurality of applications; after receipt of the request, determine whether said one of the plurality of applications complies with a predetermined set of rules; and based on the determination, selectively grant access to the requester to said one of the plurality of applications. The predetermined rules provide a guide for determining whether said one of the plurality of applications accesses a predetermined region of the memory. The processor may be further configured to authenticate an identity of the requester and grant access to said one of the plurality of applications based on the identity.

The processor may be also configured to interact with the terminal via the communicator to authenticate an identity; determine if the identity has been authenticated; and based on the determination, selectively allow communication between the terminal and the integrated circuit card.

The communicator and the terminal may communicate via communication channels. The processor may also be configured to assign one of the communication channels to the identity when the processor allows the communication between the terminal and the integrated circuit card. The processor may also be configured to assign a session key to the assigned communication channel and use the session key when the processor and the terminal communicate via the assigned communication channel.

The terminal may have a card reader, and the communicator may include a contact for communicating with the card reader. The terminal may have a wireless communication device, and the communicator may include a wireless transceiver for communicating with the wireless communication device. The terminal may have a wireless communication device, and the communicator may include a wireless transmitter for communicating with the wireless communication device.

In general, in another aspect, the invention features a method for use with an integrated circuit card and a terminal. The method includes storing an interpreter and at least one application having a high level programming language format in a memory of the integrated circuit card. A processor of the integrated circuit card uses the interpreter to interpret the at least one application for execution, and the processor uses a communicator of the card when communicating between the processor and the terminal.

In general, in another aspect, the invention features a smart card. The smart card includes a memory that stores a Java interpreter and a processor that is configured to use the interpreter to interpret a Java application for execution.

In general, in another aspect, the invention features a microcontroller that has a semiconductor substrate and a memory located in the substrate. A programming language interpreter is stored in the memory and is configured to implement security checks. A central processing unit is located in the substrate and is coupled to the memory.

Implementations of the invention may include one or more of the following. The interpreter may be a Java byte code interpreter. The security checks may include establishing firewalls and may include enforcing a sandbox security model.

In general, in another aspect, the invention features a smart card that has a programming language interpreter stored in a memory of the card. The interpreter is configured to implement security check. A central processing unit of the card is coupled to the memory.

In general, in another aspect, the invention features an integrated circuit card that is used with a terminal. The card includes a communicator and a memory that stores an interpreter and first instructions of a first application. The first instructions have been converted from second instructions of a second application. The integrated circuit card includes a processor that is coupled to the memory and is configured to use the interpreter to execute the first instructions and to communicate with the terminal via the communicator.

Implementations of the invention may include one or more of the following. The first and/or second applications may have class file format(s). The first and/or second applications may include byte codes, such as Java byte codes. The first instructions may be generalized or renumbered versions of the second instructions. The second instructions may include constant references, and the first instructions may include constants that replace the constant references of the second instructions. The second instructions may include references, and the references may shift location during the conversion of the second instructions to the first instructions. The first instructions may be relinked to the references after the shifting. The first instructions may include byte codes for a first type of virtual machine, and the second instructions may include byte codes for a second type of virtual machine. The first type is different from the second type.

In general, in another aspect, the invention features a method for use with an integrated circuit card. The method includes converting second instructions of a second application to first instructions of a first application; storing the first instructions in a memory of the integrated circuit card; and using an interpreter of the integrated circuit card to execute the first instructions.

In general, in another aspect, the invention features an integrated circuit for use with a terminal. The integrated circuit card has a communicator that is configured to communicate with the terminal and a memory that stores a first application that has been processed from a second application having a string of characters. The string of characters are represented in the first application by an identifier. The integrated circuit card includes a processor that is coupled to the memory. The processor is configured to use the interpreter to interpret the first application for execution and to use the communicator to communicate with the terminal.

In general, in another aspect, the invention features a method for use with an integrated circuit card and a terminal. The method includes processing a second application to create a first application. The second application has a string of characters. The string of characters is represented by an identifier in the second application. An interpreter and the first application are stored in a memory of the integrated circuit card. A processor uses an interpreter to interpret the first application for execution.

In general, in another aspect, the invention features a microcontroller that includes a memory which stores an application and an interpreter. The application has a class file format. A processor of the microcontroller is coupled to the memory and is configured to use the interpreter to interpret the application for execution.

In implementations of the invention, the microcontroller may also include a communicator that is configured to communicate with a terminal.

In general, in another aspect, the invention features a method for use with an integrated circuit card. The method includes storing a first application in a memory of the integrated circuit card, storing a second application in the memory of the integrated circuit card, and creating a firewall that isolates the first and second applications so that the second application cannot access either the first application or data associated with the first application.

In general, in another aspect, the invention features an integrated circuit card for use with a terminal. The integrated circuit card includes a communicator that is configured to communicate with the terminal, a memory and a processor. The memory stores applications, and each application has a high level programming language format. The memory also stores an interpreter. The processor is coupled to the memory and is configured to: a.) use the interpreter to interpret the applications for execution, b.) use the interpreter to create a firewall to isolate the applications from each other, and c.) use the communicator to communicate with the terminal.

Other advantages and features will become apparent from the following description and from the claims.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a block diagram of an integrated card system.

FIG. 2 is a flow diagram illustrating the preparation of Java applications to be downloaded to an integrated circuit card.

FIG. 3 is a block diagram of the files used and generated by the card class file converter.

FIG. 4 is a block diagram illustrating the transformation of application class file(s) into a card class file.

FIG. 5 is a flow diagram illustrating the working of the class file converter.

FIG. 6 is a flow diagram illustrating the modification of the byte codes.

FIG. 7 is a block diagram illustrating the transformation of specific byte codes into general byte codes.

FIG. 8 is a block diagram illustrating the replacement of constant references with constants.

FIG. 9 is a block diagram illustrating the replacement of references with their updated values.

FIG. 10 is a block diagram illustrating renumbering of original byte codes.

FIG. 11 is a block diagram illustrating translation of original byte codes for a different virtual machine architecture.

FIG. 12 is a block diagram illustrating loading applications into an integrated circuit card.

FIG. 13 is a block diagram illustrating executing applications in an integrated circuit card.

FIG. 14 is a schematic diagram illustrating memory organization for ROM, RAM and EEPROM.

FIG. 15 is a flow diagram illustrating the overall architecture of the Card Java virtual machine.

FIG. 16 is a flow diagram illustrating method execution in the Card Java virtual machine with the security checks.

FIG. 17 is a flow diagram illustrating byte code execution in the Card Java virtual machine.

FIG. 18 is a flow diagram illustrating method execution in the Card Java virtual machine without the security checks.

FIG. 19 is a block diagram illustrating the association between card applications and identities.

FIG. 20 is a block diagram illustrating the access rights of a specific running application.

FIG. 21 is a perspective view of a microcontroller on a smart card.

FIG. 22 is a perspective view of a microcontroller on a telephone.

FIG. 23 is a perspective view of a microcontroller on a key ring.

FIG. 24 is a perspective view of a microcontroller on a ring.

FIG. 25 is a perspective view of a microcontroller on a circuit card of an automobile.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Referring to FIG. 1, an integrated circuit card **10** (e.g., a smart card) is constructed to provide a high level, Java-based, multiple application programming and execution environment. The integrated circuit card **10** has a communicator **12***a* that is configured to communicate with a terminal communicator **12***b* of a terminal **14**. In some embodiments, the integrated circuit card **10** is a smart card with an 8 bit microcontroller, 512 bytes of RAM, 4K bytes of EEPROM, and 20K of ROM; the terminal communicator **12***b* is a conventional contact smart card reader; and the terminal **14** is a conventional personal computer running the Windows NT operating system supporting the personal computer smart card (PC/SC) standard and providing Java development support.

In some embodiments, the microcontroller, memory and communicator are embedded in a plastic card that has substantially the same dimensions as a typical credit card. In other embodiments, the microcontroller, memory and communicator are mounted within bases other than a plastic card, such as jewelry (e.g., watches, rings or bracelets), automotive equipment, telecommunication equipment (e.g., subscriber identity module (SIM) cards), security devices (e.g., cryptographic modules) and appliances.

The terminal **14** prepares and downloads Java applications to the integrated circuit card **10** using the terminal communicator **12***b*. The terminal communicator **12***b* is a communications device capable of establishing a communications channel between the integrated circuit card **10** and the terminal **14**. Some communication options include contact card readers, wireless communications via radio frequency or infrared techniques, serial communication protocols, packet communication protocols, ISO 7816 communication protocol, to name a few.

The terminal **14** can also interact with applications running in the integrated circuit card **10**. In some cases, different terminals may be used for these purposes. For example, one kind of terminal may be used to prepare applications, different terminals could be used to download the applications, and yet other terminals could be used to run the various applications. Terminals can be automated teller machines (ATMs), point-of-sale terminals, door security systems, toll payment systems, access control systems, or any other system that communicates with an integrated circuit card or microcontroller.

The integrated circuit card **10** contains a card Java virtual machine (Card JVM) **16**, which is used to interpret applications which are contained on the card **10**.

Referring to FIG. 2, the Java application **20** includes three Java source code files A.java **20***a*, B.java **20***b*, and C.java **20***c*. These source code files are prepared and compiled in a Java application development environment **22**. When the Java application **20** is compiled by the development environment **22**, application class files **24** are produced, with these class files A.class **24***a*, B.class **24***b*, and C.class **24***c* corresponding to their respective class Java source code **20***a*, **20***b*, and **20***c*. The application class files **24** follow the standard class file format as documented in chapter 4 of the Java virtual machine specification by Tim Lindholm and Frank Yellin, "The Java Virtual Machine Specification," Addison-Wesley, 1996. These application class files **24** are fed into the card class file converter **26**, which consolidates and compresses the files, producing a single card class file **27**. The card class file **27** is loaded to the integrated circuit card **10** using a conventional card loader **28**.

Referring to FIG. 3, the card class file converter **26** is a class file postprocessor that processes a set of class files **24** that are encoded in the standard Java class file format, optionally using a string to ID input map file **30** to produce a Java card class file **27** in a card class file format. One such card class file format is described in Appendix A which is hereby incorporated by reference. In addition, in some embodiments, the card class file converter **26** produces a string to ID output map file **32** that is used as input for a subsequent execution of the card class file converter.

In some embodiments, in order for the string to ID mapping to be consistent with a previously generated card class file (in the case where multiple class files reference the same strings), the card class file converter **26** can accept previously defined string to ID mappings from a string to ID input map file **30**. In the absence of such a file, the IDs are generated by the card class file converter **26**. Appendix B, which is hereby incorporated by reference, describes one possible way of implementing and producing the string to ID input map file **30** and string to ID output map file **32** and illustrates this mapping via an example.

Referring to FIG. 4, a typical application class file **24***a* includes class file information **41**; a class constant pool **42**; class, fields created, interfaces referenced, and method information **43**; and various attribute information **44**, as detailed in aforementioned Java Virtual Machine Specification. Note that much of the attribute information **44** is not needed for

this embodiment and is eliminated 45 by the card class file converter 26. Eliminated attributes include SourceFile, ConstantValue, Exceptions, LineNumberTable, LocalVariableTable, and any optional vendor attributes. The typical card class file 27 as described in Appendix A is derived from the application class files 24 in the following manner. The card class file information 46 is derived from the aggregate class file information 41 of all application class files 24*a*, 24*b*, and 24*c*. The card class file constant pool 47 is derived from the aggregate class constant pool 42 of all application class files 24*a*, 24*b*, and 24*c*. The card class, fields created, interfaces referenced, and method information 48 is derived from the aggregate class, fields created, interfaces referenced, and method information 43 of all application class files 24*a*, 24*b*, and 24*c*. The card attribute information 49 in this embodiment is derived from only the code attribute of the aggregate attribute information 44 of all application class files 24*a*, 24*b*, and 24*c*.

To avoid dynamic linking in the card, all the information that is distributed across several Java class files 24*a*, 24*b*, and 24*c* that form the application 24, are coalesced into one card class file 27 by the process shown in the flowchart in FIG. 5. The first class file to be processed is selected 51*a*. The constant pool 42 is compacted 51*b* in the following manner. All objects, classes, fields, methods referenced in a Java class file 24*a* are identified by using strings in the constant pool 42 of the class file 24*a*. The card class file converter 26 compacts the constant pool 42 found in the Java class file 24*a* into an optimized version. This compaction is achieved by mapping all the strings found in the class file constant pool 42 into integers (the size of which is microcontroller architecture dependent). These integers are also referred to as IDs. Each ID uniquely identifies a particular object, class, field or method in the application 20. Therefore, the card class file converter 26 replaces the strings in the Java class file constant pool 42 with its corresponding unique ID. Appendix B shows an example application HelloSmartCard.java, with a table below illustrating the IDs corresponding to the strings found in the constant pool of the class file for this application. The IDs used for this example are 16-bit unsigned integers.

Next, the card class file converter 26 checks for unsupported features 51*c* in the Code attribute of the input Java class file 24*a*. The Card JVM 16 only supports a subset of the full Java byte codes as described in Appendix C, which is hereby incorporated by reference. Hence, the card class file converter 26 checks for unsupported byte codes in the Code attribute of the Java class file 24*a*. If any unsupported byte codes are found 52, the card class file converter flags an error and stops conversion 53. The program code fragment marked "A" in APPENDIX D shows how these spurious byte codes are apprehended. Another level of checking can be performed by requiring the standard Java development environment 22 to compile the application 20 with a '-g' flag. Based on the aforementioned Java virtual machine specification, this option requires the Java compiler to place information about the variables used in a Java application 20 in the LocalVariableTable attribute of the class file 24*a*. The card class file converter 26 uses this information to check if the Java class file 24*a* references data types not supported by the Java card.

Next, the card class file converter 26 discards all the unnecessary parts 51*c* of the Java class file 24*a* not required for interpretation. A Java class file 24*a* stores information pertaining to the byte codes in the class file in the Attributes section 44 of the Java class file. Attributes that are not required for interpretation by the card JVM 16, such as SourceFile, ConstantValue, Exceptions, LineNumberTable, and LocalvariableTable may be safely discarded 45. The only attribute that is retained is the Code attribute. The Code attribute contains the byte codes that correspond to the methods in the Java class file 24*a*.

Modifying the byte codes 54 involves examining the Code attribute information 44 for each method in the class file, and modifying the operands of byte codes that refer to entries in the Java class file constant pool 42 to reflect the entries in the card class file constant pool 47. In some embodiments, the byte codes are also modified, as described below.

Modifying the byte codes 54 involves five passes (with two optional passes) as described by the flowchart in FIG. 6. The original byte codes 60 are found in the Code attribute 44 of the Java class file 24*a* being processed. The first pass 61 records all the jumps and their destinations in the original byte codes. During later byte code translation, some single byte code may be translated to dual or triple bytes. FIG. 7 illustrates an example wherein byte code ILOAD_0 is replaced with two bytes, byte code ILOAD and argument 0. When this is done, the code size changes, requiring adjustment of any jump destinations which are affected. Therefore, before these transformations are made, the original byte codes 60 are analyzed for any jump byte codes and a note made of their position and current destination. The program code fragment marked "B" in Appendix D shows how these jumps are recorded. Appendix D is hereby incorporated by reference.

Once the jumps are recorded, if the optional byte code translation is not being performed 62, the card class file converter 26 may proceed to the third pass 64.

Otherwise, the card class file converter converts specific byte codes into generic byte codes. Typically, the translated byte codes are not interpreted in the Card JVM 16 but are supported by converting the byte codes into equivalent byte codes that can be interpreted by the Card JVM 16 (see FIG. 7). The byte codes 70 may be replaced with another semantically equivalent but different byte codes 72. This generally entails the translation of short single specific byte codes such as ILOAD_0 into their more general versions. For example, ILOAD_0 may be replaced by byte code ILOAD with an argument 0. This translation is done to reduce the number of byte codes translated by the Card JVM 16, consequently reducing the complexity and code space requirements for the Card JVM 16. The program code fragment marked "C" in Appendix D shows how these translations are made. Note that such translations increase the size of the resulting byte code and force the re-computation of any jumps which are affected.

In the third pass 64, the card class file converter rebuilds constant references via elimination of the strings used to denote these constants. FIG. 8 shows an example wherein the byte code LDC 80 referring to constant "18" found via an index in the Java class file 24*a* constant pool 42 may be translated into BIPUSH byte code 82. In this pass the card class file converter 26 modifies the operands to all the byte codes that refer to entries in the Java class file constant pool 42 to reflect their new location in the card class file constant pool 47. FIG. 9 shows an example wherein the argument to a byte code, INVOKESTATIC 90, refers to an entry in the Java class file constant pool 42 that is modified to reflect the new location of that entry in the card class file constant pool 47. The modified operand 94 shows this transformation. The program code fragment marked "D" in Appendix D shows how these modifications are made.

Once the constant references are relinked, if the optional byte code modification is not being performed, the card class file converter may proceed to the fifth and final pass 67.

Otherwise, the card class file converter modifies the original byte codes into a different set of byte codes supported by the particular Card JVM 16 being used. One potential modification renumbers the original byte codes 60 into Card JVM 16 byte codes (see FIG. 10). This renumbering causes the byte codes 100 in the original byte codes 60 to be modified into a renumbered byte codes 102. Byte code ILOAD recognized by value 21 may be renumbered to be recognized by value 50. This modification may be done for optimizing the type tests (also known in prior art as Pass 3 checks) in the Card JVM 16. The program code fragment marked "E" in Appendix D shows an implementation of this embodiment. This modification may be done in order to reduce the program space required by the Card JVM 16 to interpret the byte code. Essentially this modification regroups the byte codes into Card JVM 16 byte codes so that byte codes with similar operands, results are grouped together, and there are no gaps between Card JVM 16 byte codes. This allows the Card JVM 16 to efficiently check Card JVM 16 byte codes and validate types as it executes.

In some embodiments, the card class file converter modifies the original byte codes 60 into a different set of byte codes designed for a different virtual machine architecture, as shown in FIG. 11. The Java byte code ILOAD 112 intended for use on a word stack 114 may be replaced by Card JVM 16 byte code ILOAD_B 116 to be used on a byte stack 118. An element in a word stack 114 requires allocating 4 bytes of stack space, whereas an element in the byte stack 118 requires only one byte of stack space. Although this option may provide an increase in execution speed, it risks losing the security features available in the original byte codes.

Since the previous steps 63, 64 or 66 may have changed the size of the byte codes 60 the card class file converter 26 has to relink 67 any jumps which have been effected. Since the jumps were recorded in the first step 61 of the card class file converter 26, this adjustment is carried out by fixing the jump destinations to their appropriate values. The program code fragment marked "F" in Appendix D shows how these jumps are fixed.

The card class file converter now has modified byte codes 68 that is equivalent to the original byte codes 60 ready for loading. The translation from the Java class file 24*a* to the card class file 27 is now complete.

Referring back to FIG. 5, if more class files 24 remain to be processed 55 the previous steps 51*a*, 51*b*, 51*c*, 52 and 54 are repeated for each remaining class file. The card class file converter 26 gathers 56 the maps and modified byte codes for the classes 24 that have been processed, places them as an aggregate and generates 57 a card class file 27. If required, the card class file converter 26 generates a string to ID output map file 32, that contains a list of all the new IDs allocated for the strings encountered in the constant pool 42 of the Java class files 24 during the translation.

Referring to FIG. 12, the card loader 28 within the terminal 14 sends a card class file to the loading and execution control 120 within the integrated circuit card 10 using standard ISO 7816 commands. The loading and execution control 120 with a card operating system 122, which provides the necessary system resources, including support for a card file system 124, which can be used to store several card applications 126. Many conventional card loaders are written in low level languages, supported by the card operating system 122. In the preferred embodiment, the bootstrap loader is written in Java, and the integrated circuit card 10 includes a Java virtual machine to run this application. A Java implementation of the loading and execution control 120 is illustrated in Appendix E which is hereby incorporated by reference. The loading and execution control 120 receives the card class file 26 and produces a Java card application 126*x* stored in the card file system 126 in the EEPROM of the integrated circuit card 10. Multiple Java card applications 126*x*, 126*y*, and 126*z* can be stored in a single card in this manner. The loading and execution control 120 supports commands whereby the terminal 14 can select which Java card application to run immediately, or upon the next card reset.

Referring to FIG. 13, upon receiving a reset or an execution command from the loading and execution control 120, the Card Java Virtual Machine (Card JVM) 16 begins execution at a predetermined method (for example, main) of the selected class in the selected Java Card application 126*z*. The Card JVM 16 provides the Java card application 126*z* access to the underlying card operating system 122, which provides capabilities such as I/O, EEPROM support, file systems, access control, and other system functions using native Java methods as illustrated in Appendix F which is hereby incorporated by reference.

The selected Java card application 126*z* communicates with an appropriate application in the terminal 14 using the communicator 12*a* to establish a communication channel to the terminal 14. Data from the communicator 12*a* to the terminal 14 passes through a communicator driver 132 in the terminal, which is specifically written to handle the communications protocol used by the communicator 12*a*. The data then passes to an integrated circuit card driver 134, which is specifically written to address the capabilities of the particular integrated circuit card 10 being used, and provides high level software services to the terminal application 136. In the preferred embodiment, this driver would be appropriate PC/SC Smartcard Service Provider (SSP) software. The data then passes to the terminal application 136, which must handle the capabilities provided by the particular card application 126*z* being run. In this manner, commands and responses pass back and forth between the terminal application 136 and the selected card application 126*z*. The terminal application interacts with the user, receiving commands from the user, some of which are passed to the selected Java card application 126*z*, and receiving responses from the Java card application 126*z*, which are processed and passed back to the user.

Referring to FIG. 14, the Card JVM 16 is an interpreter that interprets a card application 126*x*. The memory resources in the microcontroller that impact the Card JVM 16 are the Card ROM 140, Card RAM 141 and the Card EEPROM 142. The Card ROM 140 is used to store the Card JVM 16 and the card operating system 122. Card ROM 140 may also be used to store fixed card applications 140*a* and class libraries 140*b*. Loadable applications 141*a*, 141*b* and libraries 141*c* may also be stored in Card RAM 141. The Card JVM 16 interprets a card application 141*a*, 141*b*, or 140*a*. The Card JVM 16 uses the Card RAM to store the VM stack 144*a* and system state variables 144*b*. The Card JVM 16 keeps track of the operations performed via the VM stack 144*a*. The objects created by the Card JVM 16 are either on the RAM heap 144*c*, in the EEPROM heap 146*a*, or in the file system 147.

All of the heap manipulated by the Card JVM 16 may be stored in the Card RAM 141 as a RAM Heap 144*c*, or it may be distributed across to the Card EEPROM 142 as a

EEPROM Heap **146a**. Card RAM **141** is also used for recording the state of the system stack **148** that is used by routines written in the native code of the microcontroller. The Card JVM **16** uses the Card EEPROM **142** to store application data either in the EEPROM heap **146a** or in the file system **147**. Application data stored in a file may be manipulated via an interface to the card operating system **122**. This interface is provided by a class library **140b** stored in Card ROM **140**, by a loadable class library **141c** stored in Card EEPROM **142**. One such interface is described in Appendix F. Applications and data in the card are isolated by a firewall mechanism **149**.

To cope with the limited resources available on microcontrollers, the Card JVM **16** implements a strict subset of the Java programming language. Consequently, a Java application **20** compiles into a class file that contains a strict subset of Java byte codes. This enables application programmers to program in this strict subset of Java and still maintain compatibility with existing Java Virtual Machines. The semantics of the Java byte codes interpreted by the Card JVM **16** are described in the aforementioned Java Virtual Machine Specification. The subset of byte codes interpreted by the Card JVM **16** can be found in Appendix C. The card class file converter **26** checks the Java application **20** to ensure use of only the features available in this subset and converts into a form that is understood and interpreted by the Card JVM **16**.

In other embodiments, the Card JVM **16** is designed to interpret a different set or augmented set of byte codes **116**. Although a different byte code set might lead to some performance improvements, departing from a strict Java subset may not be desirable from the point of view of security that is present in the original Java byte codes or compatibility with mainstream Java development tools.

All Card JVM **16** applications **126** have a defined entry point denoted by a class and a method in the class. This entry point is mapped in the string to ID input map **30** and assigned by the card class file converter **26**. Classes, methods and fields within a Java application **20** are assigned IDs by the card class file converter **26**. For example, the ID corresponding to the main application class may be defined as **F001** and the ID corresponding to its main method, such as "main()V" could be defined as **F002**.

The overall execution architecture of the Card JVM is described by the flowchart in FIG. **15**. Execution of the Card JVM **16** begins at the execution control **120**, which chooses a card application **126z** to execute. It proceeds by finding and assigning an entry point **152** (a method) in this card application for the Card JVM **16** to interpret. The Card JVM **16** interprets the method **153**. If the interpretation proceeds successfully **154**, the Card JVM **16** reports success **155** returning control back to the execution control **120**. If in the course of interpretation **153** the Card JVM **16** encounters an unhandled error or exception (typically a resource limitation or a security violation), the Card JVM **16** stops **156** and reports the appropriate error to the terminal **14**.

An essential part of the Card JVM **16** is a subroutine that handles the execution of the byte codes. This subroutine is described by the flowchart in FIG. **16**. Given a method **160** it executes the byte codes in this method. The subroutine starts by preparing for the parameters of this method **161**. This involves setting the VM stack **144a** pointer, VM stack **144a** frame limits, and setting the program counter to the first byte code of the method.

Next, the method flags are checked **162**. If the method is flagged native, then the method is actually a call to native method code (subroutine written in the microcontroller's native processor code). In this case, the Card JVM **16** prepares for an efficient call **163** and return to the native code subroutine. The parameters to the native method may be passed on the VM stack **144a** or via the System stack **148**. The appropriate security checks are made and the native method subroutine is called. On return, the result (if any) of the native method subroutine is placed on the VM stack **144a** so that it may be accessed by the next byte code to be executed.

The dispatch loop **164** of the Card JVM **16** is then entered. The byte code dispatch loop is responsible for preparing, executing, and retiring each byte code. The loop terminates when it finishes interpreting the byte codes in the method **160**, or when the Card JVM **16** encounters a resource limitation or a security violation.

If a previous byte code caused a branch to be taken **165** the Card JVM prepares for the branch **165a**. The next byte code is retrieved **165b**. In order to keep the cost of processing each byte code down, as many common elements such as the byte code arguments, length, type are extracted and stored.

To provide the security offered by the security model of the programming language, byte codes in the class file must be verified and determined conformant to this model. These checks are typically carried out in prior art by a program referred to as the byte code verifier, which operates in four passes as described in the Java Virtual Machine Specification. To offer the run-time security that is guaranteed by the byte code verifier, the Card JVM **16** must perform the checks that pertain to the Pass 3 and Pass 4 of the verifier. This checking can be bypassed by the Card JVM **16** if it can be guaranteed (which is almost impossible to do) that the byte codes **60** interpreted by the Card JVM **16** are secure. At the minimum, code security can be maintained as long as object references cannot be faked and the VM stack **144a** and local variable bounds are observed. This requires checking the state of the VM stack **144a** with respect to the byte code being executed.

To enforce the security model of the programming language, a 256-byte table is created as shown in Appendix G which is hereby incorporated by reference. This table is indexed by the byte code number. This table contains the type and length information associated with the indexing byte code. It is encoded with the first 5 bits representing type, and the last 3 bits representing length. The type and length of the byte code is indexed directly from the table by the byte code number. This type and length is then used for checking as shown in Appendix H which is hereby incorporated by reference. In Appendix H, the checking process begins by decoding the length and type from the table in Appendix G which is hereby incorporated by reference. The length is used to increment the program counter. The type is used first for pre-execution checking, to insure that the data types on the VM stack **144a** are correct for the byte code that is about to be executed. The 256 bytes of ROM for table storage allows the original Java byte codes to be run in the Card JVM **16** and minimizes the changes required to the Java class file to be loaded in the card. Additional Java byte codes can be easily supported since it is relatively easy to update the appropriate table entries.

In other embodiments, as shown in FIG. **10**, the Java byte codes in the method are renumbered in such a manner that the byte code type and length information stored in the table in Appendix H is implicit in the reordering. Appendix H is hereby incorporated by reference. Consequently, the checks

that must be performed on the state of the VM stack **144***a* and the byte code being processed does not have to involve a table look up. The checks can be performed by set of simple comparisons as shown in Appendix I which is hereby incorporated by reference. This embodiment is preferable when ROM space is at a premium, since it eliminates a 256-byte table. However adding new byte codes to the set of supported byte codes has to be carefully thought out since the new byte codes have to fit in the implicit numbering scheme of the supported byte codes.

In another embodiment, the Card JVM **16** chooses not to perform any security checks in favor of Card JVM **16** execution speed. This is illustrated in the flowchart in FIG. **18**. The flow chart in FIG. **18** is the same as that of FIG. **16** with the security checks removed. This option is not desirable from the point of view of security, unless it can be guaranteed that the byte codes are secure.

The Card JVM **16** may enforce other security checks as well. If the byte code may reference a local variable, the Card JVM **16** checks if this reference is valid, throwing an error if it is not. If the reference is valid, the Card JVM **16** stores the type of the local variable for future checking. The VM stack **144***a* pointer is checked to see if it is still in a valid range. If not an exception is thrown. The byte code number is checked. If it is not supported, an exception is thrown.

Finally, the byte code itself is dispatched **165***d*. The byte codes translated by the Card JVM **16** are listed in Appendix C. The semantics of the byte codes are described in the aforementioned Java Virtual Machine Specification with regard to the state of the VM stack **144***a* before and after the dispatch of the byte code. Note also that some byte codes (the byte codes, INVOKESTATIC, INVOKESPECIAL, INVOKENONVIRTUAL and INVOKEVIRTUAL) may cause reentry into the Card JVM **16**, requiring processing to begin at the entry of the subroutine **161**. FIG. **17** shows the flowchart of the byte code execution routine. The routine is given a byte code **171** to execute. The Card JVM **16** executes **172** the instructions required for the byte code. If in the course of executing the Card JVM **16** encounters a resource limitation **173**, it returns an error **156**. This error is returned to the terminal **16** by the Card JVM **16**. If the byte code executes successfully, it returns a success **175**.

After execution, the type of the result is used to set the VM stack **144***a* state correctly **165***e*, properly flagging the data types on the VM stack **144***a*. The byte code information gathered previously **165***b* from the byte code info table is used to set the state of the VM stack **144***a* in accordance with the byte code that just executed.

In other embodiments, setting the output state of the VM stack **144***a* with respect to the byte code executed is simplified if the byte code is renumbered. This is shown in Appendix I which is hereby incorporated by reference.

In yet another embodiment, the Card JVM **16** may bypass setting the output state of the VM stack **144***a* in favor of Card JVM **16** execution speed. This option is not desirable from the point of view of security, unless it can be guaranteed that the byte codes are secure.

After the byte code has been executed, the byte code is retired **165***f*. This involves popping arguments off the VM stack **144***a*. Once byte code processing is completed, the loop **164** is repeated for the next byte code for the method.

Once the dispatch loop **164** terminates, the VM stack **144***a* is emptied **166**. This prevents any object references filtering down to other Card JVM **16** invocations and breaking the Card JVM's **16** security. Termination **167** of the byte code dispatch loop **164** indicates that the Card JVM **16** has completed executing the requested method.

To isolate data and applications in the integrated circuit card **10** from each other, the integrated circuit card **10** relies on the firewall mechanism **149** provided by the Card JVM **16**. Because the Card JVM implements the standard pass **3** and pass **4** verifier checks, it detects any attempt by an application to reference the data or code space used by another application, and flag a security error **156**. For example, conventional low level applications can cast non-reference data types into references, thereby enabling access to unauthorized memory space, and violating security. With this invention, such an attempt by a card application **126***z* to use a non-reference data type as a reference will trigger a security violation **156**. In conventional Java, this protected application environment is referred to as the sandbox application-interpretation environment.

However, these firewall facilities do not work independently. In fact, the facilities are overlapping and mutually reinforcing with conventional access control lists and encryption mechanisms shown in the following table:

	Acess Control Lists	Virtual Machine	Encryption
Data Protection	access control before operation	access only to own namespace	data to another program encrypted
Program Protection	access control before execution	execution only on correct types	data encrypted in program's namespace
Communication Protection	access control on channels	channel controls in own namespace	only mutually authenticated parties can communicate

Taken together, these facilities isolate both data and applications on the integrated circuit card **10** and ensure that each card application **126** can access only the authorized resources of the integrated circuit card **10**.

Referring to FIG. **19**, card applications **126***x*, **126***y*, **126***z* can be endowed with specific privileges when the card applications **126** execute. These privileges determine, for example, which data files the card applications **126** can access and what operations the card applications **126** can perform on the file system **147**. The privileges granted to the card applications **126** are normally set at the time that a particular card application **126***z* is started by the user, typically from the terminal **14**.

The integrated circuit card **10** uses cryptographic identification verification methods to associate an identity **190** (e.g., identities **190***a*, **190***b* and **190***c*) and hence, a set of privileges to the execution of the card application **126**. The association of the specific identity **190***c* to the card application **126***z* is made when the card application **126***z* begins execution, thus creating a specific running application **200**, as shown in FIG. **20**. The identity **190** is a unique legible text string reliably associated with an identity token. The identity token (e.g., a personal identification number (PIN) or a RSA private key) is an encryption key.

Referring to FIG. **20**, in order to run a specific card application **126***z*, the identity **190***c* of the card application **126***z* must be authenticated. The identity **190***c* is authenticated by demonstrating knowledge of the identity token associated with the identity **190***c*. Therefore, in order to run the card application **126***z*, an agent (e.g., a card holder or

another application wishing to run the application) must show that it possesses or knows the application's identity-defining encryption key.

One way to demonstrate possession of an encryption key is simply to expose the key itself. PIN verification is an example of this form of authentication. Another way to demonstrate the possession of an encryption key without actually exposing the key itself is to show the ability to encrypt or decrypt plain text with the key.

Thus, a specific running application **200** on the integrated circuit card **10** includes a card application **126***z* plus an authenticated identity **190***c*. No card application **126** can be run without both of these elements being in place. The card application **126***z* defines data processing operations to be performed, and the authenticated identity **190***c* determines on what computational objects those operations may be performed. For example, a specific application **126***z* can only access identity C's files **202** in the file system **147** associated with the specific identity **190***c*, and the specific card application **126***z* cannot access other files **204** that are associated with identities other than the specific identity **190***c*.

The integrated circuit card **10** may take additional steps to ensure application and data isolation. The integrated circuit card **10** furnishes three software features sets: authenticated-identity access control lists; a Java-based virtual machine; and one-time session encryption keys to protect data files, application execution, and communication channels, respectively. Collectively, for one embodiment, these features sets provide the application data firewalls **149** for one embodiment. The following discusses each software feature set and then shows how the three sets work together to insure application and data isolation on the integrated circuit card **10**.

An access control list (ACL) is associated with every computational object (e.g., a data file or a communication channel) on the integrated circuit card **10** that is be protected, i.e., to which access is to be controlled. An entry on an ACL (for a particular computational object) is in a data format referred to as an e-tuple:

type:identity:permissions

The type field indicates the type of the following identity (in the identity field), e.g., a user (e.g., "John Smith"), or a group. The permissions field indicates a list of operations (e.g., read, append and update) that can be performed by the identity on the computational object.

As an example, for a data file that has the ACL entry:

USER:AcmeAirlines:RAU,

any application whose identity is "AcmeAirlines" can read ("R"), append ("A") and update ("U") the data file. In addition, the ACL may be used selectively to permit the creation and deletion of data files. Furthermore, the ACL may be used selectively to permit execution of an application.

Whenever a computational object is accessed by a running application **200**, the access is intercepted by the Card JVM **16** and passed to the card operating system **122**, which determines if there is an ACL associated with the object. If there is an associated ACL, then the identity **190***c* associated with the running application **200** is matched on the ACL. If the identity is not found or if the identity is not permitted for the type of access that is being requested, then the access is denied. Otherwise, the access is allowed to proceed.

Referring to FIG. **13**, to prevent the potential problems due to the single data path between the integrated circuit card **10** and the terminal **14**, communication channel isolation is accomplished by including in the identity authentication process the exchange of a one-time session key **209** between the a card application **126***z* and the terminal application **136**. The key **209** is then used to encrypt subsequent traffic between the authenticating terminal application **136** and the authenticated card application **126***z*. Given the one-time session key **209**, a rogue terminal application can neither "listen in" on an authenticated communication between the terminal **14** and the integrated circuit card **10**, nor can the rogue terminal application "spoof" the card application into performing unauthorized operations on its behalf.

Encryption and decryption of card/terminal traffic can be handled either by the card operating system **122** or by the card application itself **126***z*. In the former case, the communication with the terminal **14** is being encrypted transparently to the application, and message traffic arrives decrypted in the data space of the application. In the latter case, the card application **126***z* elects to perform encryption and decryption to provide an extra layer of security since the application could encrypt data as soon as it was created and would decrypt data only when it was about to be used. Otherwise, the data would remain encrypted with the session key **209**.

Thus, the application firewall includes three mutually reinforcing software sets. Data files are protected by authenticated-identity access control lists. Application execution spaces are protected by the Card JVM **16**. Communication channels are protected with one-time session encryption keys **209**.

In other embodiments, the above-described techniques are used with a microcontroller (such as the processor **12**) may control devices (e.g., part of an automobile engine) other than an integrated circuit card. In these applications, the microcontroller provides a small platform (i.e., a central processing unit, and a memory, both of which are located on a semiconductor substrate) for storing and executing high level programming languages. Most existing devices and new designs that utilize a microcontroller could use this invention to provide the ability to program the microcontroller using a high level language, and application of this invention to such devices is specifically included.

The term application includes any program, such as Java applications, Java applets, Java aglets, Java servlets, Java commlets, Java components, and other non-Java programs that can result in class files as described below.

Class files may have a source other than Java program files. Several programming languages other than Java also have compilers or assemblers for generating class files from their respective source files. For example, the programming language Eiffel can be used to generate class files using Pirmin Kalberer's "J-Eiffel", an Eiffel compiler with JVM byte code generation (web site: http://www.spin.ch/~kalberer/jive/index.htm). An Ada **95** to Java byte code translator is described in the following reference (incorporated herein by reference): Taft, S. Tucker, "Programming the Internet in Ada 95", proceedings of Ada Europe '96, 1996. Jasmin is a Java byte code assembler that can be used to generate class files, as described in the following reference (incorporated herein by reference): Meyer, Jon and Troy Downing, "Java Virtual Machine", O'Reilly, 1997. Regardless of the source of the class files, the above description applies to languages other than Java to generate codes to be interpreted.

FIG. **21** shows an integrated circuit card, or smart card, which includes a microcontroller **210** that is mounted to a plastic card **212**. The plastic card **212** has approximately the

same form factor as a typical credit card. The communicator **12a** can use a contact pad **214** to establish a communication channel, or the communicator **12a** can use a wireless communication system.

In other embodiments, a microcontroller **210** is mounted into a mobile or fixed telephone **220**, effectively adding smart card capabilities to the telephone, as shown in FIG. **22**. In these embodiments, the microcontroller **210** is mounted on a module (such as a Subscriber Identity Module (SIM)), for insertion and removal from the telephone **220**.

In other embodiments, a microcontroller **210** is added to a key ring **230** as shown in FIG. **23**. This can be used to secure access to an automobile that is equipped to recognize the identity associated with the microcontroller **210** on the key ring **230**.

Jewelry such as a watch or ring **240** can also house a microcontroller **210** in an ergonomic manner, as shown in FIG. **24**. Such embodiments typically use a wireless communication system for establishing a communication channel, and are a convenient way to implement access control with a minimum of hassle to the user.

FIG. **25** illustrates a microcontroller **210** mounted in an electrical subsystem **252** of an automobile **254**. In this embodiment, the microcontroller is used for a variety of purposes, such as to controlling access to the automobile, (e.g. checking identity or sobriety before enabling the ignition system of the automobile), paying tolls via wireless communication, or interfacing with a global positioning system (GPS) to track the location of the automobile, to name a few.

While specific embodiments of the present invention have been described, various modifications and substitutions will become apparent to one skilled in the art by this disclosure. Such modifications and substitutions are within the scope of the present invention, and are intended to be covered by the appended claims.

What is claimed is:

1. An integrated circuit card for use with a terminal, comprising:

a communicator configured to communicate with the terminal;

a memory storing:

an application derived from a program written in a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including at least one step selected from a group consisting of

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa;

modifying byte code operands from references using identifying strings to references using unique identifiers; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

an interpreter operable to interpret such an application derived from a program written in a high level programming language format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the application for execution and to use the communicator to communicate with the terminal.

2. The integrated circuit card of claim **1**, wherein the high level programming language format comprises a class file format.

3. The integrated circuit card of claim **1** wherein the processor comprises a microcontroller.

4. The integrated circuit card of claim **1** wherein at least a portion of the memory is located in the processor.

5. The integrated circuit card of claim **1** wherein the high level programming language format comprises a Java programming language format.

6. The integrated circuit card of claim **1**, wherein

the application has been processed from a second application having a plurality of program elements, at least one being a string of characters, and

wherein in the first application the string of characters is replaced with an identifier.

7. The integrated circuit card of claim **6**, wherein the identifier comprises an integer.

8. The integrated circuit card of claim **1** wherein the processor is further configured to:

receive a request from a requester to access an element of the card;

after receipt of the request, interact with the requester to authenticate an identity of the requester; and

based on the identity, selectively grant access to the element.

9. The integrated circuit card of claim **8**, wherein the requester comprises the processor.

10. The integrated circuit card of claim **8**, wherein the requester comprises the terminal.

11. The integrated circuit card of claim **8**, wherein

the element comprises the application stored in the memory, and

once access is allowed, the requester is configured to use the application.

12. The integrated circuit card of claim **8**, wherein

the element comprises another application stored in the memory.

13. The integrated circuit card of claim **8**, wherein the element includes data stored in the memory.

14. The integrated circuit card of claim **8** wherein the element comprises the communicator.

15. The integrated circuit card of claim **8**, wherein the memory also stores an access control list for the element, the access control list furnishing an indication of types of access to be granted to the identity, the processor further configured to:

based on the access control list, selectively grant specific types of access to the requester.

16. The integrated circuit card of claim **15** wherein the types of access include reading data.

17. The integrated circuit card of claim **15** wherein the types of access include writing data.

18. The integrated circuit card of claim **15** wherein the types of access include appending data.

19. The integrated circuit card of claim **15** wherein the types of access include creating data.

20. The integrated circuit card of claim **15** wherein the types of access include deleting data.

21. The integrated circuit card of claim **15** wherein the types of access include executing an application.

22. The integrated circuit card of claim **1**, wherein the application is one of a plurality of applications stored in the memory, the processor is further configured to:

receive a request from a requester to access one of the plurality of applications;

after receipt of the request, determine whether said one of the plurality of applications complies with a predetermined set of rules; and

based on the determination, selectively grant access to the requester to said one of the plurality of applications.

23. The integrated circuit card of claim 22, wherein the predetermined rules provide a guide for determining whether said one of the plurality of applications accesses a predetermined region of the memory.

24. The integrated circuit card of claim 22, wherein the processor is further configured to:

authenticate an identity of the requester; and

grant access to said one of the plurality of applications based on the identity.

25. The integrated circuit card of claim 1, wherein the processor is further configured to:

interact with the terminal via the communicator to authenticate an identity; and

determine if the identity has been authenticated; and

based on the determination, selectively allow communication between the terminal and the integrated circuit card.

26. The integrated circuit card of claim 25, wherein the communicator and the terminal communicate via communication channels, the processor further configured to assign one of the communication channels to the identity when the processor allows the communication between the terminal and the integrated circuit card.

27. The integrated circuit card of claim 26, wherein the processor is further configured to:

assign a session key to said one of the communication channels, and

use the session key when the processor and the terminal communicate via said one of the communication channels.

28. The integrated circuit card of claim 1, wherein the terminal has a card reader and the communicator comprises a contact for communicating with the card reader.

29. The integrated circuit card of claim 1, wherein the terminal has a wireless communication device and the communicator a wireless transceiver for communicating with the wireless communication device.

30. The integrated circuit card of claim 1, wherein the terminal has a wireless communication device and the communicator comprises a wireless transmitter for communicating with the wireless communication device.

31. A method for use with an integrated circuit card and a terminal, comprising:

storing an interpreter operable to interpret programs derived from programs written in a high level programming language and an application derived from a program written in a high level programming language format in a memory of the integrated circuit card wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including at least one step selected from a group consisting of

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa;

modifying byte code operands from references using identifying strings to references using unique identifiers; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

using a processor of the integrated circuit card to use the interpreter to interpret the application for execution; and

using a communicator of the card when communicating between the processor and the terminal.

32. The method of claim 31, wherein the high level programming language format comprises a class file format.

33. The method of claim 31, wherein the processor comprises a microcontroller.

34. The method of claim 31, wherein at least a portion of the memory is located in the processor.

35. The method of claim 31, wherein the high level programming language format comprises a Java programming language format.

36. The method of claim 31, wherein

the application has been processed from a second application having a plurality of program elements, at least one being a string of characters, further comprising:

replacing the string of characters in the first application with an identifier.

37. The method of claim 36, wherein the identifier includes an integer.

38. The method of claim 31, further comprising:

receiving a request from a requester to access an element of the card;

after receipt of the request, interacting with the requester to authenticate an identity of the requester; and

based on the identity, selectively granting access to the element.

39. The method of claim 38, wherein the requester comprises the processor.

40. The method of claim 38, wherein the requester comprises the terminal.

41. The method of claim 38, wherein the element comprises the application stored in the memory, further comprising:

once access is allowed, using the application with the requester.

42. The method of claim 38, wherein the element comprises another application stored in the memory.

43. The method of claim 38, wherein the element includes data stored in the memory.

44. The method of claim 38, wherein the element comprises the communicator.

45. The method of claim 38, wherein the memory also stores an access control list for the element, the access control list furnishing an indication of types of access to be granted to the identity, further comprising:

based on the access control list, using the processor to selectively grant specific types of access to the requester.

46. The method of claim 45, wherein the types of access include reading data.

47. The method of claim 45, wherein the types of access include writing data.

48. The method of claim 45, wherein the types of access include appending data.

49. The method of claim 45, wherein the types of access include creating data.

50. The method of claim 45, wherein the types of access include deleting data.

51. The method of claim 45, wherein the types of access including executing an application.

52. The method of claim 31, wherein the application is one of a plurality of applications stored in the memory, further comprising:

receiving a request from a requester to access one of the applications stored in the memory;

upon receipt of the request, determining whether said one of the plurality of applications complies with a predetermined set of rules; and

based on the determining, selectively granting access to the said one of the plurality of applications.

53. The method of claim 52, wherein the predetermined rules provide a guide for determining whether said one of the plurality of applications accesses a predetermined region of the memory.

54. The method of claim 52, further comprising:

authenticating an indentity of the requester; and

based on the indentity, granting access to said one of the plurality of applications.

55. The method of claim 31, further comprising:

communicating with the terminal to authenticate an identity;

determining if the identity has been authenticated; and

based on the determining, selectively allowing communication between the terminal and the integrated circuit card.

56. The method of claim 55, further comprising:

communicating between the terminal and the processor via communication channels; and

assigning one of the communication channels to the identity when the allowing allows communication between the card reader and the integrated circuit card.

57. The method of claim 56, further comprising:

assigning a session key to said one of the communication channels; and

using the session key when the processor and the terminal communicate via said one of the communication channels.

58. A microcontroller comprising:

a memory storing:

a derivative application derived from an application having a class file format wherein the application is derived from an application having a class file format by first compiling the application having a class file format into a compiled form and then converting the compiled form into a converted form, the converting step including at least one step selected from a group consisting of

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa;

modifying byte code operands from references using identifying strings to references using unique identifiers; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation, and

an interpreter configured to interpret applications derived from applications having a class file format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the derivative application for execution.

59. The microcontroller of claim 58, further comprising:

a communicator configured to communicate with a terminal.

60. The microcontroller of claim 59, wherein the terminal has a card reader and the communicator comprises a contact for communicating with the card reader.

61. The microcontroller of claim 59, wherein the terminal has a wireless communicator and a wireless transceiver for communicating with the wireless communication device.

62. The microcontroller of claim 59, wherein the terminal has a wireless communication device and the communicator comprises a wireless transmitter for communicating with the wireless communication device.

63. The microcontroller of claim 58, wherein the class file format comprises a Java class file format.

64. An integrated circuit card for use with a terminal, comprising:

a communicator configured to communicate with the terminal;

a memory storing:

applications, each application derived from applications having a high level programming language format, and

an interpreter operable to interpret applications derived from applications having a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including at least one step selected from a group consisting of

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa;

modifying byte code operands from references using identifying strings to references using unique identifiers; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

a processor coupled to the memory, the processor configured to:

a.) use the interpreter to interpret the applications for execution,

b.) use the interpreter to create a firewall to isolate the applications from each other, and

c.) use the communicator to communicate with the terminal.

65. A microcontroller having a set of resource constraints and comprising:

a memory, and

an interpreter loaded in memory and operable within the set of resource constraints, the microcontroller having: at least one application loaded in the memory to be interpreted by the interpreter, wherein the at least one application is generated by a programming environment comprising:

a) a compiler for compiling application source programs written in high level language source code form into a compiled form, and

b) a converter for post processing the compiled form into a minimized form suitable for interpretation within the set of resource constraints by the interpreter, wherein the converter comprises means for translating from the byte codes in the compiled form to byte codes in a format suitable for interpretation by the interpreter by:

a) using at least one step in a process including the steps:

a.1) recording all jumps and their destinations in the original byte codes;

a.2) converting specific byte codes into equivalent generic byte codes or vice-versa;

a.3) modifying byte code operands from references using identifying strings to references using unique identifiers; and

a.4) renumbering byte codes in the compiled form to equivalent byte codes in the format suitable for interpretation; and

b) relinking jumps for which destination address is effected by conversion step a.1, a.2, a.3, or a.4.

66. The microcontroller of claim **65**, wherein the compiled form includes attributes, and the converter comprises a means for including attributes required by the interpreter while not including the attributes not required by the interpreter.

67. The microcontroller of claim **65** wherein the compiled form is in a standard Java class file format and the converter accepts as input the compiled form in the standard Java class file format and produces output in a form suitable for interpretation by the interpreter.

68. The microcontroller of claim **65** wherein the compiled form includes associating an identifying string for objects, classes, fields, or methods, and the converter comprises a means for mapping such strings to unique identifiers.

69. The microcontroller of claim **68** wherein each unique identifier is an integer.

70. The microcontroller of claim **68** wherein the mapping of strings to unique identifiers is stored in a string to identifier map file.

71. The microcontroller of claim **65** where in the high level language supports a first set of features and a first set of data types and the interpreter supports a subset of the first set of features and a subset of the first set of data types, and wherein the converter verifies that the compiled form only contains features in the subset of the first set of features and only contains data types in the subset of the first set of data types.

72. The microcontroller of claim **65** wherein the application program is compiled into a compiled form for which resources required to execute or interpret the compiled form exceed those available on the microcontroller.

73. The microcontroller of claim **65** wherein the compiled form is designed for portability on different computer platforms.

74. The microcontroller of claim **65** wherein the interpreter is further configured to determine, during an interpretation of an application, whether the application meets a security criteria selected from a set of rules containing at least one rule selected from the set:

not allowing the application access to unauthorized portions of memory,

not allowing the application access to unauthorized microcontroller resources,

wherein the application is composed of byte codes and checking a plurality of byte codes at least once prior to execution to verify that execution of the byte codes does not violate a security constraint.

75. The microcontroller of claim **65** wherein at least one application program is generated by a process including the steps of:

prior to loading the application verifying that the application does not violate any security constraints; and

loading the application in a secure manner.

76. The microcontroller of claim **75** wherein the step of loading in a secure manner comprises the step of:

verifying that the loading identity has permission to load applications onto the microcontroller.

77. The microcontroller of claim **75** wherein the step of loading in a secure manner comprises the step of:

encrypting the application to be loaded using a loading key.

78. A method of programming a microcontroller having a memory and a processor operating according to a set of resource constraints, the method comprising the steps of:

inputting an application program in a first programming language;

compiling the application program in the first programming language into a first intermediate code associated with the first programming language, wherein the first intermediate code being interpretable by at least one first intermediate code virtual machine;

converting the first intermediate code into a second intermediate code, wherein the step of converting comprises:

at least one of the steps of:

a) recording all jumps and their destinations in the original byte codes;

b) converting specific byte codes into equivalent generic byte codes or vice-versa;

c) modifying byte code operands from references using identifying strings to references using unique identifiers; and

d) renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

relinking jumps for which destination address is effected by conversion step a), b), c), or d);

wherein the second intermediate code is interpretable within the set of resource constraints by at least one second intermediate code virtual machine; and

loading the second intermediate code into the memory of the microcontroller.

79. The method of programming a microcontroller of claim **78** wherein the step of converting further comprises:

associating an identifying string for objects, classes, fields, or methods; and

mapping such strings to unique identifiers.

80. The method of claim **79** wherein the step of mapping comprises the step of mapping strings to integers.

81. The method of claim **80** wherein the step of loading the second intermediate code into the memory of the microcontroller further comprises checking the second intermediate code prior to loading the second intermediate code to verify that the second intermediate code meets a predefined integrity check and that loading is performed according to a security protocol.

82. The method of claim **81** wherein the security protocol requires that a particular identity must be validated to permit loading prior to the loading of the second intermediate code.

83. The method of claim **81** further characterized by providing a decryption key and wherein the security protocol requires that the second intermediate code is encrypted using a loading key corresponding to the decryption key.

84. An integrated circuit card for use with a terminal, comprising:

a communicator configured to communicate with the terminal;

a memory storing:

an application derived from a program written in a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including the steps of:

modifying byte code operands from references using identifying strings to references using unique identifiers;

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

an interpreter operable to interpret such an application derived from a program written in a high level programming language format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the application for execution and to use the communicator to communicate with the terminal.

85. A method for use with an integrated circuit card and a terminal, comprising:

storing an interpreter operable to interpret programs derived from programs written in a high level programming language and an application derived from a program written in a high level programming language format in a memory of the integrated circuit card wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including:

modifying byte code operands from references using identifying strings to references using unique identifiers;

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation;

using a processor of the integrated circuit card to use the interpreter to interpret the application for execution; and

using a communicator of the card when communicating between the processor and the terminal.

86. An integrated circuit card for use with a terminal, comprising:

a communicator configured to communicate with the terminal;

a memory storing:

applications, each application derived from applications having a high level programming language format, and

an interpreter operable to interpret applications derived from applications having a high level programming language format wherein the application is derived from a program written in a high level programming language format by first compiling the program into a compiled form and then converting the compiled form into a converted form, the converting step including the steps of:

modifying byte code operands from references using identifying strings to references using unique identifiers;

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation; and

a processor coupled to the memory, the processor configured to:

a.) use the interpreter to interpret the applications for execution,

b.) use the interpreter to create a firewall to isolate the applications from each other, and

c.) use the communicator to communicate with the terminal.

87. A microcontroller comprising:

a memory storing:

a derivative application derived from an application having a class file format wherein the application is derived from an application having a class file format by first compiling the application having a class file format into a compiled form and then converting the compiled form into a converted form, the converting step including:

recording all jumps and their destinations in the original byte codes;

converting specific byte codes into equivalent generic byte codes or vice-versa; and

renumbering byte codes in a compiled format to equivalent byte codes in a format suitable for interpretation, and

an interpreter configured to interpret applications derived from applications having a class file format; and

a processor coupled to the memory, the processor configured to use the interpreter to interpret the derivative application for execution.

* * * * *

US005534857A

United States Patent [19]

Laing et al.

[11] **Patent Number: 5,534,857**

[45] **Date of Patent: Jul. 9, 1996**

[54] **METHOD AND SYSTEM FOR SECURE, DECENTRALIZED PERSONALIZATION OF SMART CARDS**

[75] Inventors: **Simon G. Laing; Matthew P. Bowcock**, both of Sydney, Australia

[73] Assignee: **Security Domain Pty. Ltd.**, Australia

[21] Appl. No.: **232,088**

[22] PCT Filed: **Nov. 10, 1992**

[86] PCT No.: **PCT/AU92/00608**

§ 371 Date: **Apr. 28, 1994**

§ 102(e) Date: **Apr. 28, 1994**

[87] PCT Pub. No.: **WO93/10509**

PCT Pub. Date: **May 27, 1993**

[30] **Foreign Application Priority Data**

Nov. 12, 1991 [AU] Australia PK9443

[51] **Int. Cl.**^{6} **G07F 7/08**; H04L 9/32

[52] **U.S. Cl.** **340/825.34**; 340/825.3; 380/21; 380/23; 235/380

[58] **Field of Search** 340/825.31, 825.34, 340/825.33; 380/23, 24, 25; 364/401, 406; 235/380, 382, 382.5, 487

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,453,074	6/1984	Weinstein	235/380
4,649,233	3/1987	Bass et al.	380/21
4,758,718	7/1988	Fujisaki	340/825.32
4,803,351	2/1989	Shigenaga	340/825.34
4,910,774	3/1990	Barakat	380/23
4,965,568	10/1990	Atalla et al.	340/825.34
5,068,894	11/1991	Hoppe	380/23
5,109,152	4/1992	Takagi et al.	325/380
5,193,114	3/1993	Moseley	380/23
5,196,840	3/1993	Leith et al.	340/825.3

FOREIGN PATENT DOCUMENTS

0374012	6/1990	European Pat. Off.	G07F 7/10

Primary Examiner—Brian Zimmerman
Assistant Examiner—William H. Wilson, Jr.
Attorney, Agent, or Firm—Dressler, Goldsmith, Shore & Milnamow, Ltd.

[57] **ABSTRACT**

A method and apparatus for securely writing confidential data from an issuerer to a customer smart card at a remote location includes, establishing a communication link between a retailer data terminal device at the remote location and the issuer's secure computer. A communication link is established between a secure terminal device, which includes a smart card reader/writer, and the data terminal device. The retailer is authenticated to the issuer and the issuer to the retailer by means of a retailer smart card presented to the secure terminal device. A session key is established for enciphering data traffic between the secure terminal device and the issuer's computer using the retailer smart card. The customer smart card is presented to the secure terminal device. Confidential customer data is enciphered using the session key and it is written from the issuer's computer to the customer smart card.

10 Claims, 4 Drawing Sheets

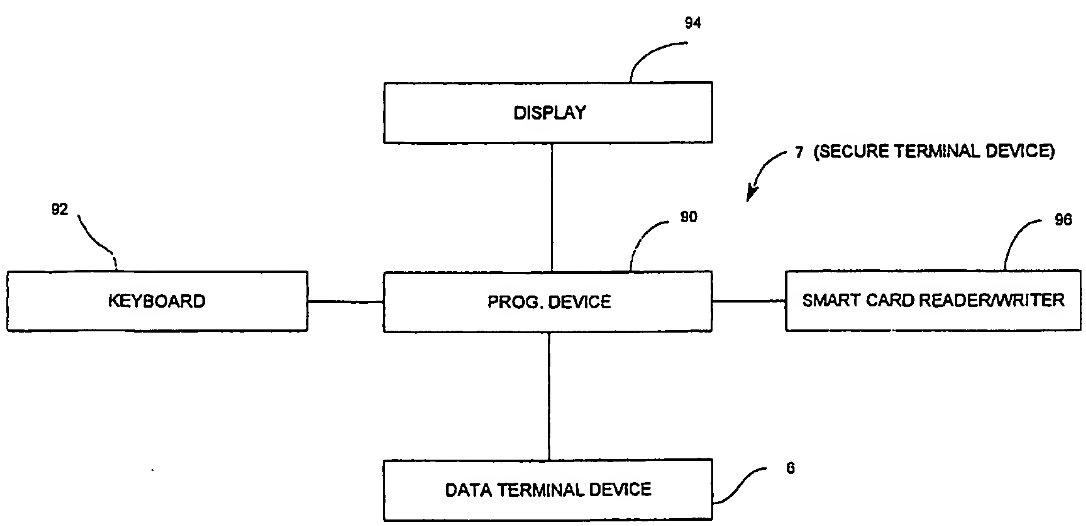
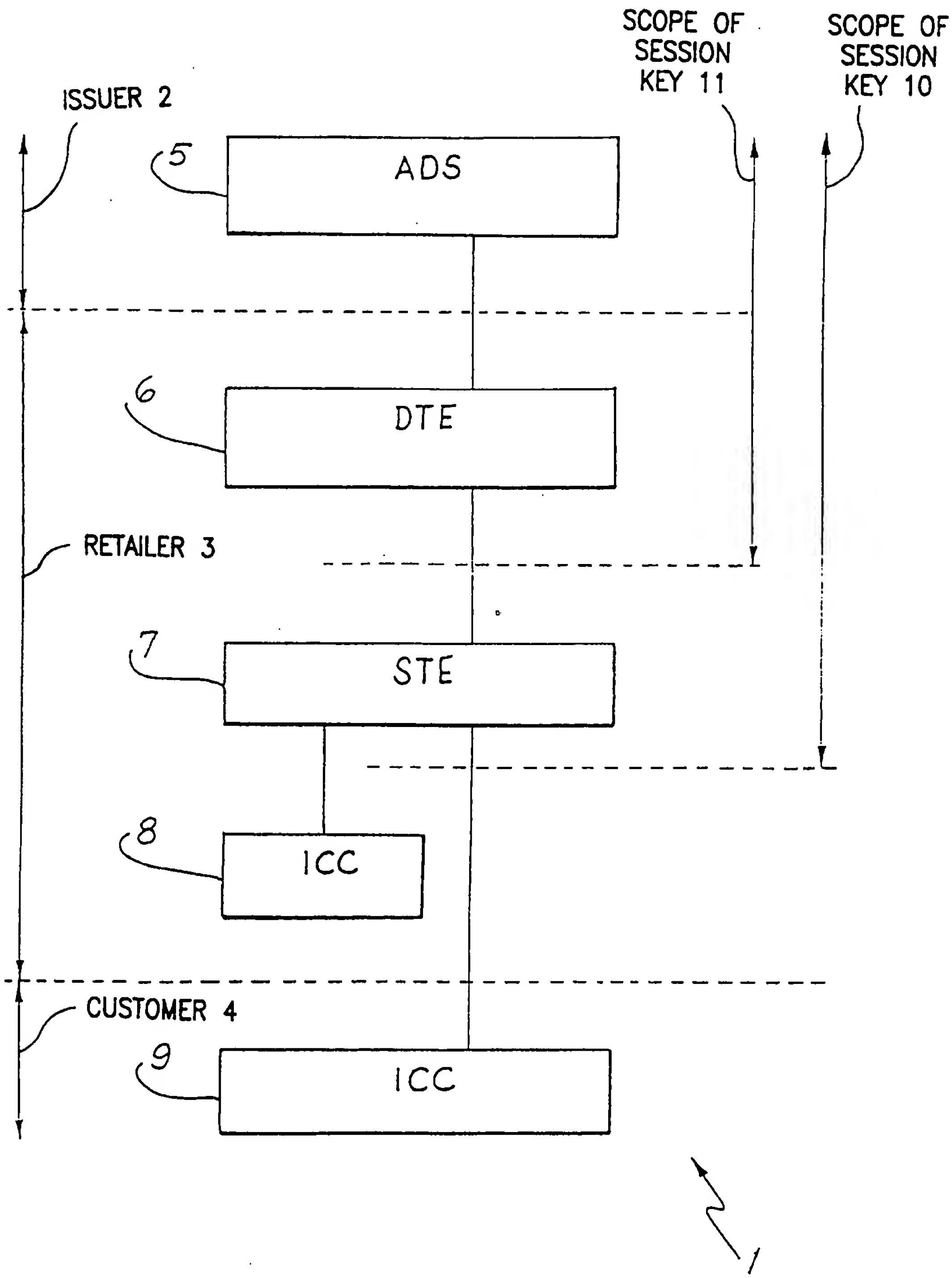


FIG. 1


SCOPE OF SESSION KEY 11
SCOPE OF SESSION KEY 10
ISSUER 2
5
ADS
6
DTE
RETAILER 3
7
STE
8
ICC
CUSTOMER 4
9
ICC
1

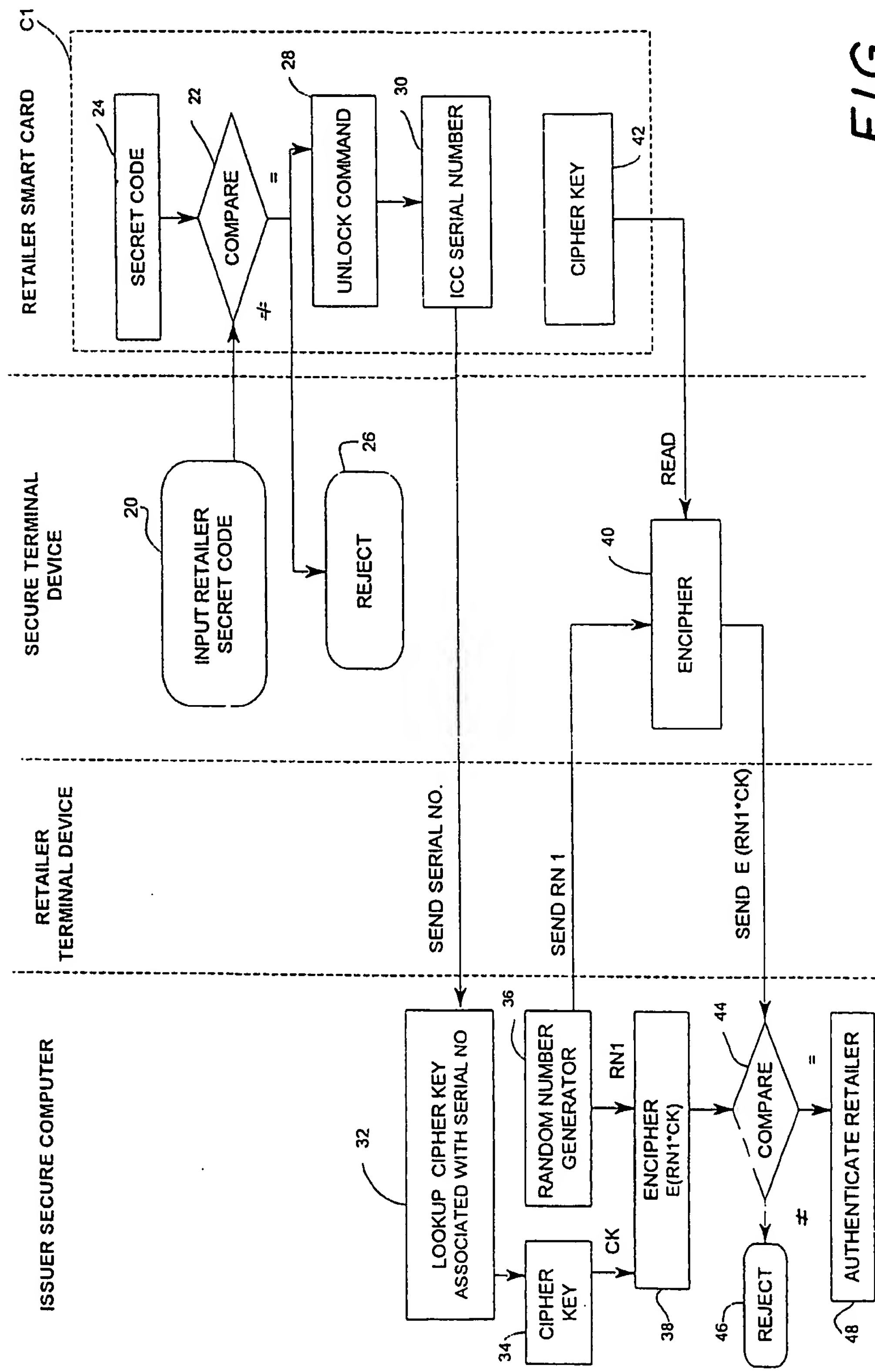
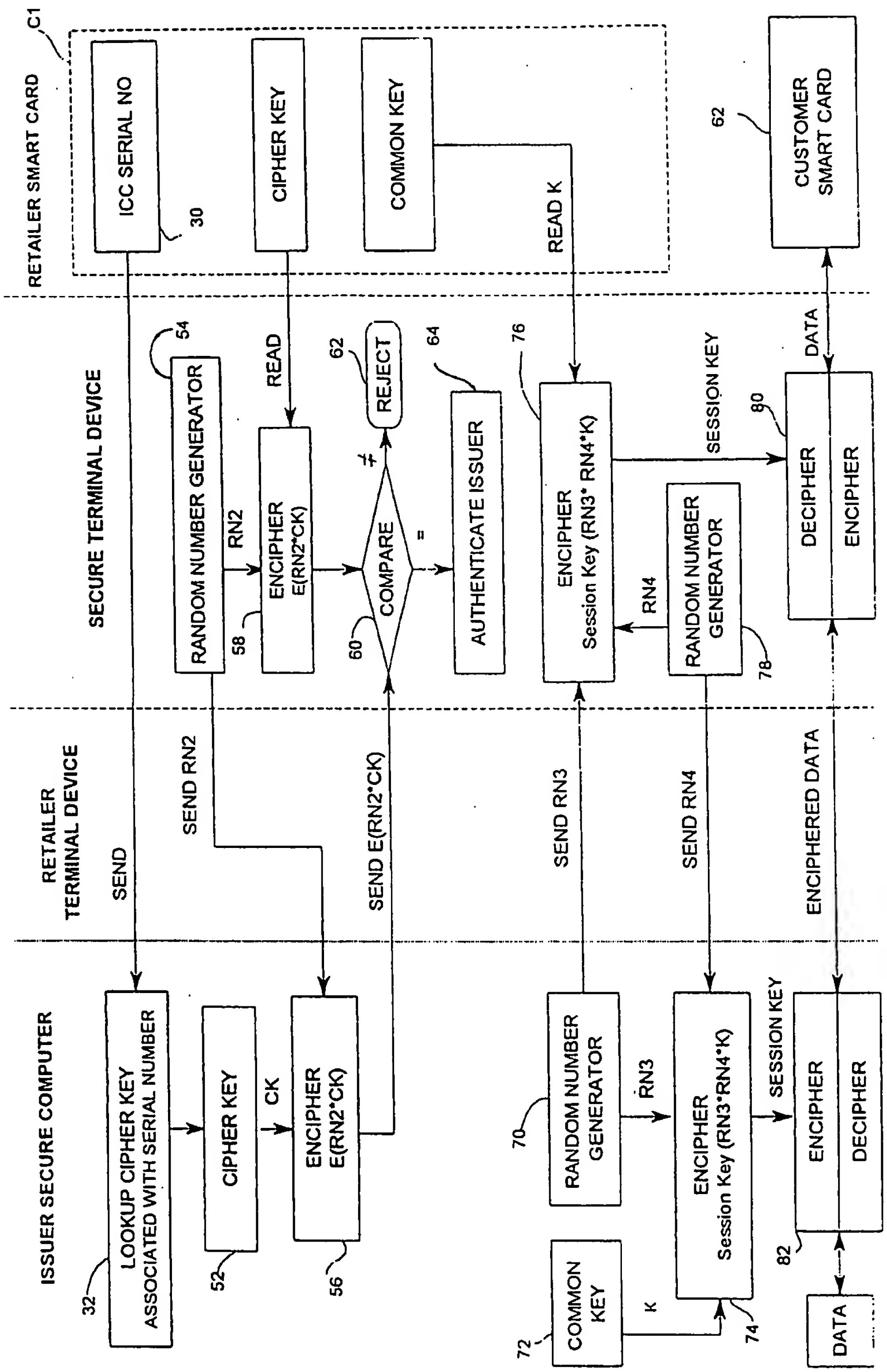

RETAILER SMART CARD
C1
SECRET CODE
24
COMPARE
22
=
≠
UNLOCK COMMAND
28
ICC SERIAL NUMBER
30
CIPHER KEY
42
SECURE TERMINAL DEVICE
INPUT RETAILER SECRET CODE
20
REJECT
26
READ
ENCIPHER
40
RETAILER TERMINAL DEVICE
SEND SERIAL NO.
SEND RN 1
SEND E (RN1*CK)
ISSUER SECURE COMPUTER
LOOKUP CIPHER KEY ASSOCIATED WITH SERIAL NO
32
CIPHER KEY
34
RANDOM NUMBER GENERATOR
36
RN1
CK
ENCIPHER E(RN1*CK)
38
COMPARE
44
=
≠
REJECT
46
AUTHENTICATE RETAILER
48

FIG. 2

FIG. 3


RETAILER SMART CARD
C1
ICC SERIAL NO
30
CIPHER KEY
COMMON KEY
READ K
CUSTOMER SMART CARD
62
SECURE TERMINAL DEVICE
RANDOM NUMBER GENERATOR
54
READ
RN2
ENCIPHER E(RN2*CK)
58
COMPARE
60
≠
=
REJECT
62
AUTHENTICATE ISSUER
64
ENCIPHER Session Key (RN3* RN4*K)
76
RN4
RANDOM NUMBER GENERATOR
78
SESSION KEY
DATA
DECIPHER
ENCIPHER
80
RETAILER TERMINAL DEVICE
SEND
SEND RN2
SEND E(RN2*CK)
SEND RN3
SEND RN4
ENCIPHERED DATA
ISSUER SECURE COMPUTER
LOOKUP CIPHER KEY ASSOCIATED WITH SERIAL NUMBER
32
CIPHER KEY
52
CK
ENCIPHER E(RN2*CK)
56
RANDOM NUMBER GENERATOR
70
RN3
ENCIPHER Session Key (RN3*RN4*K)
74
SESSION KEY
ENCIPHER
DECIPHER
82
COMMON KEY
72
K
DATA

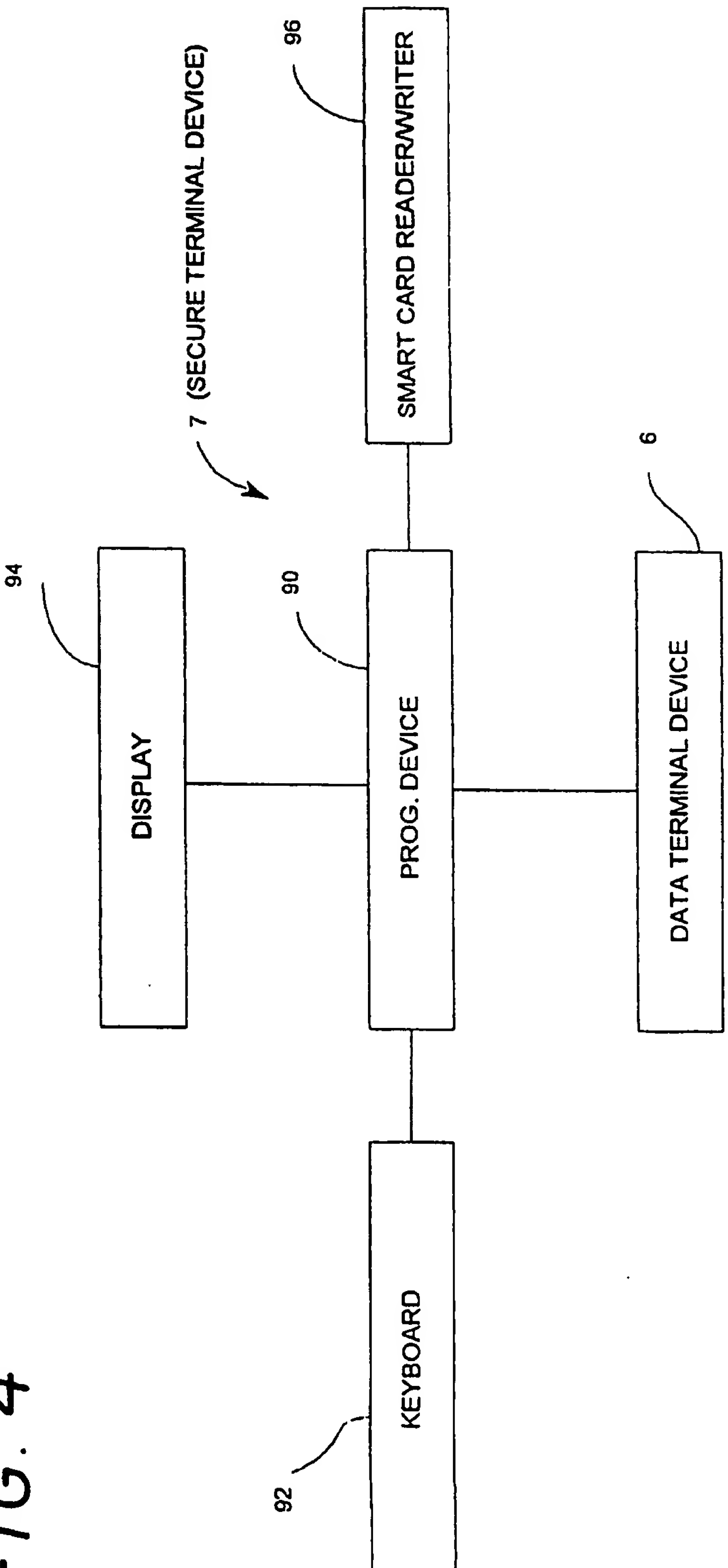

7 (SECURE TERMINAL DEVICE)
96
SMART CARD READER/WRITER
94
DISPLAY
90
PROG. DEVICE
6
DATA TERMINAL DEVICE
92
KEYBOARD

FIG. 4

METHOD AND SYSTEM FOR SECURE, DECENTRALIZED PERSONALIZATION OF SMART CARDS

TECHNICAL FIELD

This invention concerns a method for securely writing confidential data to smart cards in remote, insecure locations. In a second aspect the invention concerns a system for securely writing the confidential data. Smart Cards are used as a highly-secure means of storing data in a portable form. They are of particular use, for example, in cryptographic applications for the storage of cipher keys.

BACKGROUND OF THE INVENTION

When a smart card is manufactured, the manufacturer 'burns in' a unique identifying serial number. In addition the manufacturer installs a manufacturer's 'Master' Secret Code.

The card and the Master Secret Code are subsequently conveyed to the Issuer by separate means. Upon receipt by the Issuer the card is accessed by presenting the Master Secret Code and that code is then changed to a fresh 'Issuer' Secret Code not known to the manufacturer. One or more User Secret Codes are then stored in the card and used to protect access to confidential user data. Initial user data may then be stored in the card. The card and the User Secret Code(s) are ultimately conveyed to a user by separate means, and the appropriate User Secret Code(s) must be correctly presented to the smart card by the user, before access to the card is allowed.

The process of presentation of the Master Secret Code, storage of the Issuer Secret Code, storage of the User Secret Codes, and initial storage of user data, is commonly called Personalisation, and is traditionally done in a secure "Personalisation Centre" by the Issuer. This approach is costly, time-consuming and relatively insecure.

SUMMARY OF THE INVENTION

According to the present invention, as currently envisaged, there is provided a method for securely writing confidential data from an Issuer to a customer smart card at a remote location, comprising the steps of:

establishing a communications link between a retailer data terminal device at the remote location and the Issuer's secure computer;

establishing a communications link between a secure terminal device, which includes a smart card reader/writer, and the data terminal device;

authenticating the retailer to the Issuer and the Issuer to the retailer, by means of a retailer smart card presented to the secure terminal device;

establishing a session key for enciphering data traffic between the secure terminal device and the Issuer's computer, using the retailer smart card;

presenting the customer smart card to the secure terminal device; then

enciphering the confidential data under the session key and writing it from the Issuer's computer to the customer smart card.

Preferably the method includes the step of establishing a second session key for enciphering data traffic between the data terminal device and the Issuer's computer.

Preferably the retailer is authenticated to the Issuer by entering a retailer secret code which is checked by the retailer smart card, then a cipher key is read from the retailer smart card to the secure terminal device and checked by a challenge sent by the Issuer. Optionally the Issuer is subsequently authenticated to the retailer using a cipher key which is read from the retailer smart card to the secure terminal device and used to challenge the Issuer.

Preferably the session keys are established by using a cipher key to encrypt the combined product of two random numbers, one of which was generated by the first party and sent to the second party, the other of which was generated by the second party and sent to the first party.

Advantageously the confidential data is an Issuer Secret Code present in the customer smart card to prevent access to the card, and required to open the card to accept data.

Preferably the confidential data comprises a directory and file structures, and data.

According to a further aspect of the invention, as currently envisaged, there is provided a system for securely writing confidential data from an Issuer to a customer smart card in a remote location, comprising:

the Issuer's secure computer;

a retailer data terminal device at the remote location selectively in communication with the computer by means of a communications link;

a secure terminal device at the remote location, including a smart card reader/writer, selectively in communication with the computer via the data terminal device;

a retailer smart card containing the data required to authenticate the retailer to the Issuer and the Issuer to the retailer, and the data required to establish a session key for enciphering traffic between the secure terminal device and the Issuer's computer;

a customer smart card able to accept the confidential data, when presented to the secure terminal device, written from the computer enciphered under the session key.

Preferably the retailer smart card also contains the data required to establish a second session key for enciphering traffic between the data terminal device and the Issuer's computer.

Preferably the confidential data is an Issuer Secret Code, present in the customer smart card to prevent access to the card, and required to open the card to accept data.

This method and system permit personalisation of the smart card at a location convenient to the customer, such as the point of sale of the item, or service, with which the smart card is subsequently to be used. Such locations are unlikely to be secure, may be widely dispersed from any central administrative centre, and may be operated by staff who do not work for the Card Issuer. Furthermore the method provides a decentralised personalisation service in a manner that ensures the security of all confidential data transferred between components of the system.

As smart cards are used more widely in mass consumer applications such as mobile telephony and Pay TV, the high volume of smart cards issued, and the widely dispersed customer population will make decentralised personalisation highly cost-effective and competitive.

Once the infrastructure for a decentralised personalisation system is in place, it can be used for securely loading data other than personalisation data into previously personalised smart cards.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 is a schematic diagram showing the relationships between the components of a system according to the invention.

FIG. 2 is a schematic flow chart showing the steps of the method of writing confidential information from an issuer's secure computer to a customer smart card at a remote location up to authentication of the retailer;

FIG. 3 is a schematic flow chart showing the steps of the method of writing confidential information from an issuer's secure computer to a customer smart card at a remote location up to enciphered data transfer between the customer smart card and the secure computer; and

FIG. 4 is a block diagram of the secure terminal device STE7.

DESCRIPTION OF THE PREFERRED EMBODIMENT

Method and system 1 involve the interaction of three entities:

The Issuer 2 is the organisation which ultimately provides the goods or services that are obtained through the use of the customer smart card. It is responsible for the system as a whole, for the purchase of smart cards, and for their supply to Retailers. This organisation could be the central office of a bank, or a telecommunications operator, for example.

The Retailer 3 is the institution which represents the Issuer 2 in a particular local area. It could be a bank branch, or a newsagent, for example.

The Customer 4 is the end-user of the service, and the holder of the smart card that gives access to that service.

The elements involved in the process of decentralised personalisation are:

A Central Administration System 5 (ADS).

A computer system in a secure location that is equipped to communicate by telecommunications links with the other, remotely sited, components of the system. These links are assumed to be insecure. The system 5 also includes a secure database of Retailer Keys.

A Data Terminal Device 6 (DTD).

A small computer system (such as a Personal Computer) located in the Retailer's premises. It is equipped to communicate, by a telecommunications link, with the Central Administration System. This system is not considered to be secure by the Issuer.

A Secure Terminal Device 7 (STE).

A tamper-resistant, programmable device comprising a numeric and function keypad, a display, and a smart card reader/writer. It communicates with the Data Terminal device 6 by a serial communications link.

FIG. 4 is a block diagram of the secure terminal device STE7. That device includes a tamper-resistant programmable device 90 which in turn receives information from a key pad 92, displays information on a display 94 and is coupled to a smart card read/writer 96. It communicates with a data terminal device DTE6 via a serial communications link.

Smart Cards or Integrated Circuit Cards (ICC).

These are read and written to by the Secure Terminal device. Two categories of smart card are used within the system:

Retailer Cards 8

Each Retailer is issued with one Retailer Card, which has already been securely personalised by the Issuer. It contains the data required to gain access to, and use, the system. This data is protected from access by several Secret Codes, some known only to the Retailer, and some known only to the Central Administration System.

Customer Smart Cards 9

These are the smart cards that will be issued by the Retailer 3 to his Customers 4. They are held in stock in an unpersonalised state, exactly as they were shipped from the card manufacturer.

The operation of the method and system will be described by analysing each phase in the personalisation of a Customer smart card from the perspective of the Retailer. These phases are identified as:

Session Establishment;

Personalisation of Customer Smart Card;

Session Termination;

Modification of Data on Customer Smart Cards.

In general, there are several different operations involved in each phase.

Session Establishment

1) Retailer System Startup

On startup, the Data Terminal device sets up a communications link with the Central Administration System. This link is used for all future communications between the Central Administration System and the Data Terminal device.

2) Retailer Sign-On

Once the communications link is established, the Retailer is prompted to insert his Retailer Card in the Secure Terminal device. The Retailer is then prompted by the Secure Terminal device to enter his personal Secret Code which is passed directly to the smart card for checking.

3) Retailer Authentication

If the check of the Retailer's Secret Code succeeds, the Secure Terminal device reads a unique unprotected, read-only serial number from the smart card, and sends it to the Central Administration System via the Data Terminal device. Thus the Administration System knows which smart card is in use.

The Secure Terminal device then reads a unique cipher key out of a file on the smart card which was set up during personalisation so that it can only be read after the Retailer's Secret Code has been correctly presented.

The Central Administration. System then sends a random number (a challenge) to the Secure Terminal device, via the Data Terminal device. The Secure Terminal device enciphers the challenge using the cipher key read from the smart card and sends the result (the response) back to the Central Administration System. Since the Central Administration System maintains a record of the keys held on every Retailer Card issued, it is able to validate the response by also enciphering the random number challenge using the same cipher key, and comparing the result with the response received from the Secure Terminal device. If the two values are identical, the Retailer has successfully authenticated himself to the Central Administrative System.

With respect to FIG. 2, a retailer small card C1 is inserted into the secure terminal device. In a step 20, the retailer enters a personal security code which in a step 22 is compared to a secret code read from the retailer card C1 in a step 24. If the codes do not correspond, the terminal rejects the card C1 in a step 26. If the two codes do correspond, the terminal issues an unlock command in a step 28 and reads a unique, unprotected, read-only serial number from the card

C1 in a step 30 and transmits that number to the issuer's secure computer. In a step 32 the issuer's secure computer retrieves a cipher key 34 associated with the serial number of the card C1 and in a random number generator 36 generates a random number RN1. The random number RN1 is then enciphered in a step 38. The random number RN1 is also transmitted to the secure terminal device and is enciphered in a step 40 using a cipher key 42 carried by the smart card C1. The enciphered output from the secure terminal device is then transmitted back to the secure computer and compared in a step 44 to the output of the local enciphering step 38. If there is no match, the transaction will be rejected in a step 46. If there is a match, the retailer will be authenticated in a step 48.

4) Issuer Authentication

Authentication of the Retailer only provides part of the security needed. It is equally important to ensure that the Central Administration System is authentic. This is achieved by performing an enciphered challenge-response in the reverse direction using a random data challenge generated within the Secure Terminal device, and using a key read from the Retailer Card. If the Central Administration System is authentic, it will also have a record of this key, and will be able to encipher the challenge and send back the correct response.

5) Establishment of Session Keys

Once both the Central Administration System and the Retailer System have authenticated each other, they can mutually establish session keys for enciphering future data traffic between them. This is done by one party sending the other a random number. Both parties then combine these two numbers together (for example, by exclusive ORing them) and encipher the result, using a key known only to them, to produce a new number—the Session Key. Future data traffic can then be enciphered using this session key. Whenever the session is terminated, and a new one started, new random numbers are used, resulting in a new session key.

Two session keys are required for securing communication between the different components of the system, one 10 between the Secure Terminal device 7 and the Central Administration System 5 and a second, optional, key 11 between the Data Terminal device 6 and the Central Administration System 5. By using different session keys, tight security can be maintained because intermediate parties in an exchange of messages between two parties are not privy to the contents of the messages they are simply passing on.

6) Collection and Transmission of Customer Details

The Retailer may now obtain from the Customer any personal data required by the Central Administration System before personalisation of a Customer smart card can proceed. This data may be entered into the Data Terminal device, enciphered under the Data Terminal device-Central Administration System session key 11 (to protect the confidentiality of the Customer data in transit over the link), and sent to the Central Administration System.

7) Assessment of Customer Data

If appropriate, the Central Administration System now checks the Customer data (for example, runs a credit check), and determines whether or not personalisation of a Customer smart card may proceed. The decision is communicated to the Retailer via the Data Terminal device.

Personalisation of Customer smart card

8) Selection of Customer smart card

If the Central Administration System allows personalisation to proceed, the Retailer removes his Retailer Card from the Secure Terminal device, selects a smart card from stock, and inserts it in the Secure Terminal device. The identity of the smart card is then communicated to the Central Administration System, either by the Retailer entering identifying information into the Data Terminal device, or by the Secure Terminal device reading a Serial Number out of the smart card and sending it to the Central Administration System.

9) Presentation of Manufacturer's Master Secret Code

At this stage, the smart card is protected from general access by a unique Master Secret Code written into it by the manufacturer. The method by which the Master Secret Code can be computed for any smart card in a batch will have been separately communicated to the Card Issuer. In order to gain access to the smart card, its Master Secret Code must be presented and this is done by computing the Master Secret Code in the Central Administration System then sending it to the Secure Terminal device, enciphered under the Central Administration System-Secure Terminal device session key 10. In the Secure. Terminal device, it is deciphered and presented to the smart card. This has the effect of opening up the smart card for further accesses.

10) Smart Card Set Up

Once the smart card has been "opened" by presentation of the Master Secret Code, it can be set up to meet the Customer's and Issuer's requirements. This involves creating various data structures on the smart card, and writing appropriate data to them, and to other locations on the smart card. All instructions on the manner in which the smart card is to be set up are sent from the Central Administration System enciphered under the Central Administration System-Secure Terminal device session key 10. Similarly, all data written to the smart card are sent from the Central Administration System enciphered under the Central Administration System-Secure Terminal device session key 10.

11) Entry of Customer Secret Code

At this point, the Customer may be required to enter the Secret Code he will subsequently use to protect access to his personal data held on the smart card. He is prompted on the Secure Terminal device display to enter his Customer Secret Code, and does so using the Secure Terminal device's keypad. This ensures that nobody else, not even the Retailer, knows his Secret Code, The entered Secret Code is written to the smart card where it is securely stored to be used by the smart card microprocessor to validate future presentations of the Customer Secret Code.

With respect to FIG. 3, the issuer is first authenticated. In a step 52, at the issuer's secure computer, a cipher key associated with the serial number which had been previously received in step 32, is determined. The associated cipher key is retrieved in a step 52. The secure terminal device in a step 54 uses a random number generator to generate a random number RN2. This random number is transmitted to the issuer's secure computer and enciphered in a step 56. It is also enciphered at the secure terminal device in a step 58. The issuer's secure computer transmits the enciphered result from the step 56 to the secure terminal device which compares in a step 60 that received enciphered result to the locally generated enciphered result, from the step 58. If there is no match, the attempt at authentication of the issuer is rejected in a step 62. In the event in a step 60 the two enciphered codes match, in a step 64, the terminal authenticates the issuer. Once the issuer's secure computer has been authenticated at the secure terminal device, a session key can be established. A random number generator 70, at the issuer's secure computer, generates a random number RN3 and transmits same to the secure terminal device. Using a common key 72 associated with the retailer smart card C1 present at the issuer's secure computer, the common key and

the random number RN3 along with another random number, RN4 received from the secure terminal device, generated in a step **78**, are enciphered to produce a session key. Similarly, at the secure terminal device in a step **76**, the locally generated random number RN4 along with the received random number RN3 and the common key from the retailer smart card C1 are enciphered in the step **76** to produce the session key at the secure terminal device. As is apparent from FIG. **3**, a session key is required at the secure terminal device as well as to the issuer's secure computer. Information in steps **80**, **82** can be transmitted between the customer's smart card, C2 and the issuer's secure computer after enciphering and deciphering using the session key. This is a bidirectional data transmission.

Session Termination

12) Customer Smart Card Handover

The Customer may now remove his smart card from the Secure Terminal device and begin to use it.

13) Termination of Communications Session

The communications session with the Central Administration System is now terminated, which involves erasure of all session keys that were being used.

14) Breaking of Communications Link

The communications link with the Central Administration System may now be broken, or left open for use in the personalisation of other smart cards.

Modification of Data on Customer smart cards

There may be a need to modify some of the secure data on the Customer's smart card, at some stage after personalisation. This can be accomplished by using exactly the same method, but varying the data that is written to the Customer smart card during the "Smart Card Set Up" step.

With respect of FIG. **4**, the secure terminal device STE7 includes a tamper-resistant programmable device **90** which in turn receives information from a key pad **92**, displays information on a display **94** and is coupled to a smart card read/writer **96**. It communicates with a data terminal device DTE6 via a serial communications link.

An Example of Practical Implementation

To take a specific example, the GSM digital mobile telephone network relies upon smart cards called Subscriber Identity Modules (SIMs), inserted in mobile telephone handsets to authenticate users as valid subscribers to the network. It also subsequently uses the Subscriber Identity Module to generate a different session key for each phone call made. This session key is used to encipher all data, such as voice data, transmitted from, and to, that mobile telephone during that call. In order to operate, therefore, each Subscriber Identity Module must be individually initialised to contain unique, identifying information and cryptographic keys prior to issue to a subscriber.

Each Retailer is provided with the following:

- a Personal Computer (Data Terminal device);
- a secure, tamper-resistant PIN pad (Secure Terminal device), which incorporates a smart card reader;
- a Retailer smart card, already personalised by the Issuer and set up to contain:
  - a Retailer Secret Code known only to the Retailer;
  - cipher keys known only to the Issuer, in a file protected by an Issuer Secret Code from general access;
  - a stock of unpersonalised blank Subscriber Identity Modules, that are protected from general access by a Manufacturing Secret Code.

When a prospective new Subscriber to the network approaches the Retailer to open a subscription, the Retailer establishes a communications link with the Central Administration System, using his Retailer smart card to authenticate himself, and to authenticate the Central Administration System, and to establish session keys between the Secure Terminal device and Central Administration System, and between the Data Terminal device and Central Administration System.

The Retailer then enters the new Subscriber's personal, and financial details into the Data Terminal device, where they are enciphered using the Central Administration System-Data Terminal device session key and sent to the Central Administration System. In the Central Administration System, the details are deciphered and used to run a credit check on the new Subscriber. If this is successful, the Retailer is notified, by means of an enciphered message sent from the Central Administration System to the Data Terminal device, that personalisation can proceed.

The Retailer selects a Subscriber Identity Module from his stock, depending on Subscriber preference, and the type of mobile telephone the Subscriber will use. He inserts the Subscriber Identity Module in the Secure Terminal device and the personalisation data is sent from the Central Administration System, enciphered under the Central Administration System-Secure Terminal device session key. This data is deciphered in the Secure Terminal device before being written to the Subscriber Identity Module. This data includes instructions on the directory and file structures to be set up in the Subscriber Identity Module, as well as the information that is to be written to certain of these files, and to other locations in the Subscriber Identity Module. Data of particular note that is written to the Subscriber Identity Module at this time is:

- the Subscriber's unique International Mobile Subscriber Identification (IMSI) number;
- the authentication key (Ki);
- the Subscriber Identity Module Service Table, which defines which of the available network services the Subscriber has actually accepted;
- the PLMN Selector, which sets up an initial order of preference for the selection of network, when the Subscriber is out of range of his home network.

Once the Subscriber Identity Module has been set up, the Subscriber may enter his PIN Code (which will be his personal Secret Code protecting access to the Subscriber Identity Module) into the Secure Terminal device, which writes it to the Subscriber Identity Module. He may also enter his PIN unblocking key which is also written to the Subscriber Identity Module for use in the event the user forgets his PIN code.

The telephone number of the Subscriber is then communicated, enciphered under the Central Administration System-Data Terminal device session key, from the Central Administration System to the Data Terminal device. The Retailer informs the Subscriber of the number, prints out a record of the entire transaction, and hands the new Subscriber his Subscriber Identity Module. The Subscriber is then in a position to use the network.

At this point all communications sessions are terminated by the erasure of the session keys and the communications link may be broken.

Since all information written to the Subscriber Identity Module originated from the Central Administration System, the Central Administration System holds a complete record of what is stored on the Subscriber Identity Module, as well as personal, financial and other Subscriber information. It is therefore able to route calls to the Subscriber, allocate charges correctly as they are incurred, and issue bills.

We claim:

1. A method for securely writing confidential data from

issuer's secure computer to a customer smart card presented to a secure terminal device with smart card reader/writer connected to a retailer's data terminal device at a remote location, including the steps of:

(a) establishing a communications link between the data terminal device and the secure computer;

(b) authenticating the retailer to the issuer by:

(i) presenting a retailer smart card to the secure terminal device reader/writer and establishing access to information stored in the smart card by entering a retailer secret code into the secure terminal device to unlock the retailer smart card

(ii) reading data from the unlocked retailer smart card and sending only information pertaining to the identity of the retailer smart card to the secure computer;

(iii) generating and sending from the secure computer a first random number to the secure terminal device;

(iv) enciphering the first random number at the secure terminal device using a cipher key read from the unlocked retailer smart card, the cipher key having a value unrelated to the retailer secret code, and sending the enciphered first random number back to the secure computer;

(v) comparing the retailer smart card identification data with data stored in the secure computer to identify the retailer smart card, then retrieving a cipher key stored in the secure computer associated with the identification data and enciphering the first random number with the cipher key; and

(vi) comparing the enciphered first random number received from the secure terminal device with the enciphered first random number generated in the secure computer to authenticate the retailer when the values of the enciphered first random numbers are identical;

(c) establishing a mutual session key for enciphering data transfer between the secure terminal and the secure computer after authentication of the retailer to the issuer has been effected, the mutual session key being generated by using a common key stored in the secure computer and the retailer smart card;

(d) retrieving the retailer smart card and subsequently presenting the customer smart card to the secure terminal device;

(e) enciphering at the secure computer, the confidential data to be written to the customer smart card using the mutual session key and sending the enciphered confidential data to the secure terminal device; and

(f) deciphering at the secure terminal device, the enciphered confidential data using the mutual session key and writing the confidential data on to the customer smart card.

2. A method according to claim 1 including, after step (b), the step of

(g) authenticating the issuer to the retailer by performing an enciphered challenge-response including:

(i) generating at the secure terminal device a second random number, sending the second random number to the secure computer, and enciphering the second random number using a cipher key read from the unlocked retailer smart card;

(ii) using the identification data of the retailer smart card, for the purpose of retrieving the cipher key stored in the secure computer associated with the identification data, enciphering the second random number using the cipher key and sending: the enciphered second random number back to the secure terminal device; and

(iii) comparing the enciphered second random number received from the secure computer with the enciphered second random number generated in the secure terminal device to authenticate the issuer when the values of the enciphered second random numbers are identical.

3. A method according to claim 1 or claim 2, wherein the session key is established by the secure computer generating and sending a first random number to the secure terminal device, the secure terminal device generating a second random number and sending the second random number to the secure computer, the secure computer and the secure terminal device each enciphering the combined product of the two random numbers using the common key stored in the secure computer and the retailer smart card to generate the session key.

4. A method according to claim 1, wherein the confidential data to be written on the customer smart card is an issuer secret code which enables locking and unlocking of the customer smart card, the issuer secret code being required to unlock the card to accept data.

5. A method according to claim 4, wherein the data also comprises a directory and file structures and other consumer specific data.

6. A method according to claim 1, wherein a second session key is established for enciphering traffic between the data terminal device and the issuer's secure computer in a manner analogous to the establishment of the session key for enciphering traffic between the secure terminal device and the secure computer.

7. A system for securely writing confidential data from an issuer to a customer smart card in a remote location comprising:

an issuer's secure computer containing data pertaining to the identification of a plurality of retailer smart cards and respective associated cipher keys;

a retailer data terminal device at the remote location selectively in communication with the secure computer by means of a communications link;

a secure terminal device at the remote locating including a smart card reader/writer, selectively in communication with the secure computer via the data terminal device;

a retailer smart card containing data required to authenticate the retailer to the issuer including a retailer secret code to enable unlocking of the smart card upon positive comparison, with a secret code inputted into the secure terminal device, data pertaining to the identity of the smart card, a cipher key to encipher an authentication challenge generated by the secure computer and sent to the secure terminal device, and data required to establish a session key for enciphering traffic between the secure terminal device and the secure computer including a common cipher key stored in the retailer smart card and the secure computer; and

a customer smart card able to accept the confidential data, when presented to the secure terminal device, sent from the computer to the secure data terminal after being deciphered using the session key.

8. A secure terminal which can be coupled to a remote computer, and a data link, intended for use with first and second, different, authorization cards comprising:

a programmed processor;

an input device coupled to said processor; and

a card reader/write coupled to said processor wherein said processor includes means for reading a first indicium from a first card and a second indicium entered via said input device and for comparing same, said processor including means, responsive to said comparing for reading a third, identifying, indicium from said first card and for transmitting same to the remote computer and for receiving a random number response from the remote computer, associated with said identifying indicium, and for reading a fourth, key indicium from the first card for combining said random numeric response with said key indicium thereby producing an enciphered random numeric response sent to the remote computer for authentication, wherein said processor includes means for establishing a different transaction enciphering key in response to said authentication and wherein said processor includes means for reading a second card and for authorizing transactions using said transaction key and an identifying indicium carried by said second card and not entered by said input device.

9. A terminal as in claim 8 wherein said processor includes means for entering onto said second card a user specified identifying indicium different from said transaction enciphering key.

10. A terminal as in claim 8 wherein said processor includes means for terminating communication with the remote computer and wherein said transaction enciphering key is erased in response to said termination.

* * * * *

US006115719A

United States Patent [19]

Purdy et al.

[11] **Patent Number: 6,115,719**

[45] **Date of Patent: Sep. 5, 2000**

[54] **JAVA COMPATIBLE OBJECT ORIENTED COMPONENT DATA STRUCTURE**

[75] Inventors: **H. Cameron Purdy**, Wilmington; **Yevgeniy Gleyzer**, Lexington, both of Mass.

[73] Assignee: **Revsoft Corporation**, Andover, Mass.

[21] Appl. No.: **09/196,995**

[22] Filed: **Nov. 20, 1998**

[51] **Int. Cl.**[7] **G06F 15/18**; G06F 17/30

[52] **U.S. Cl.** **707/103**; 707/100

[58] **Field of Search** 707/103, 100, 707/102; 395/702, 703, 707

[56] **References Cited**

U.S. PATENT DOCUMENTS

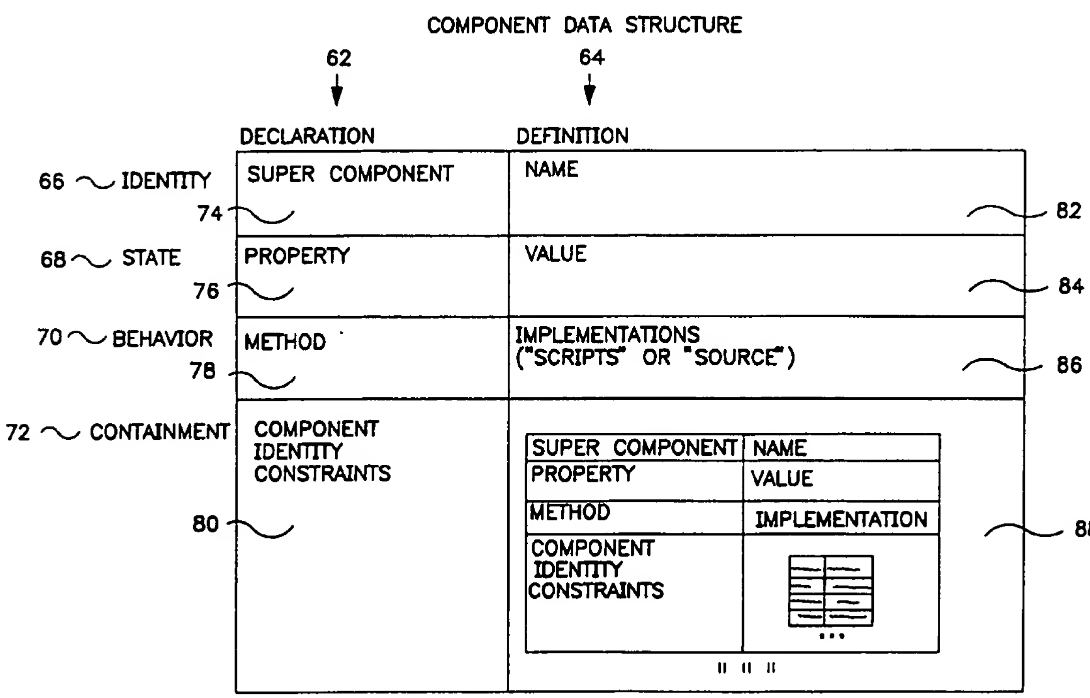
5,313,630	5/1994	Namioka et al.	395/600
5,838,965	11/1998	Kavanagh et al.	395/614
5,895,477	4/1999	Orr et al.	707/517
5,903,894	5/1999	Reneris	707/100
5,905,987	5/1999	Shutt et al.	707/103
5,918,052	6/1999	Kruskal et al.	395/701
5,953,726	9/1999	Carter et al.	707/103
5,978,582	11/1999	McDonald et al.	395/702
6,003,038	12/1999	Chen	707/103
6,018,741	1/2000	Howland et al.	707/102
6,049,665	4/2000	Branson et al.	395/702

Primary Examiner—Hosain T. Alam
Assistant Examiner—Sanjiv Shah
Attorney, Agent, or Firm—Perkins, Smith & Cohen; Jerry Cohen

[57] **ABSTRACT**

An object-oriented component data structure and method for constructing, extending, assembling, and modifying software components. The inventive component data structure provides a complete definition of a component including its identity, state, behavior, and containment of other components, which are recursively nested instances of the inventive component data structure. The inventive component data structure supports inheritance, such that the definition of a component inherits from the definition of another component; contained component data structures likewise support inheritance. Moreover, the inventive component data structure and resulting software components are compatible with the Java Virtual Machine (JVM), Java Class File, Java Byte Code, JavaBean, and CORBA specifications. The inventive component data structure and method are particularly well suited for providing tools for software development, trouble-shooting, and systems integration. Furthermore, the inventive component data structure minimizes the need for manual changes with respect to customized and localized software components when component revisions are made.

15 Claims, 6 Drawing Sheets



INHERITANCE IN JAVA

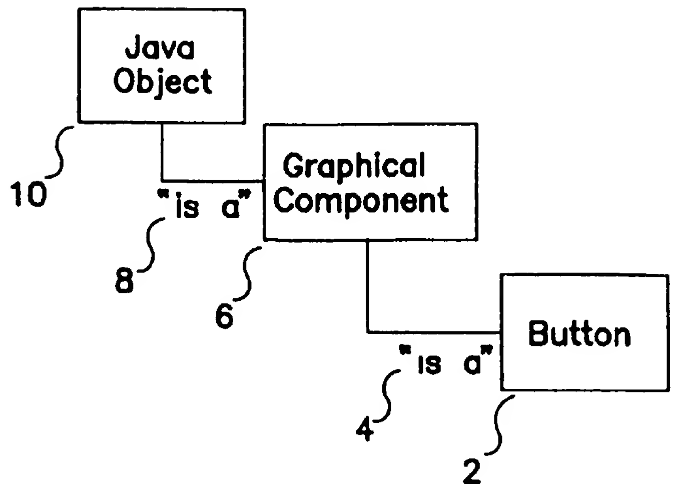


FIG. 1A

(PRIOR ART)

IHHERITANCE HIERARCHY

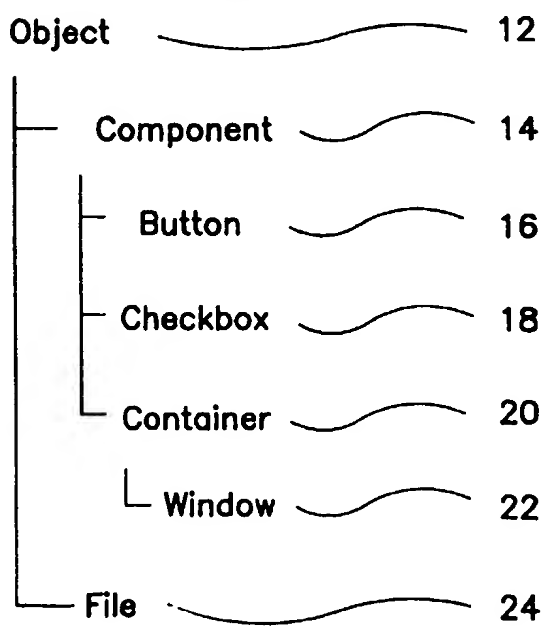


FIG. 1B

(PRIOR ART)

CONTAINMENT IN JAVA

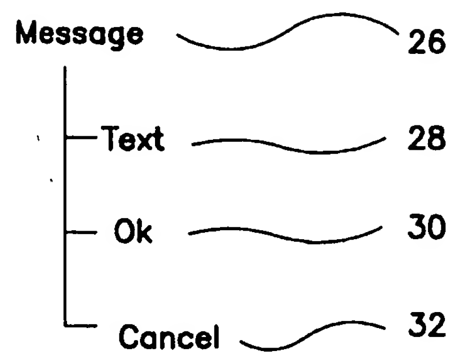


FIG. 2A
(PRIOR ART)

NESTED CONTAINMENT

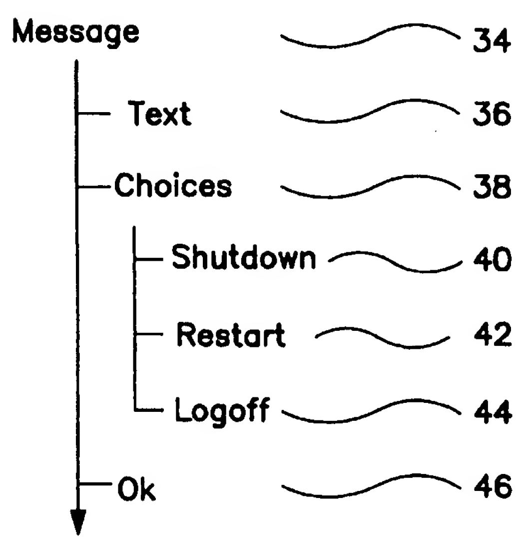


FIG. 2B
(PRIOR ART)

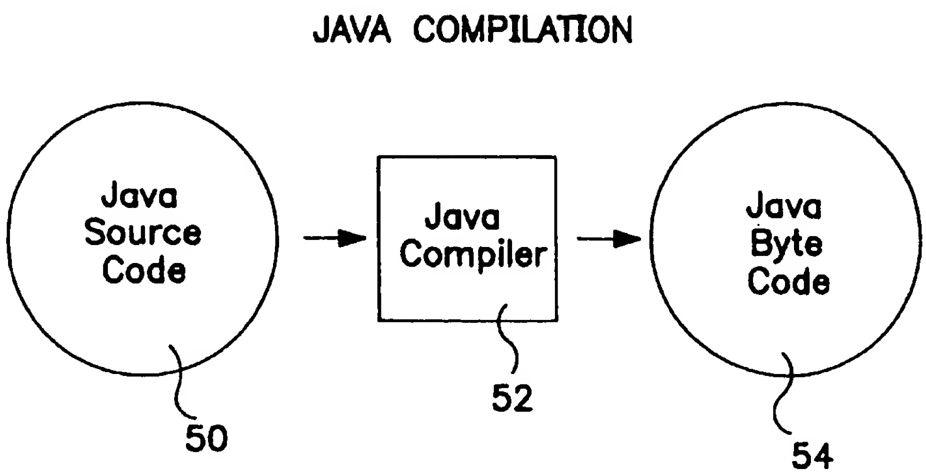


FIG. 3A
(PRIOR ART)

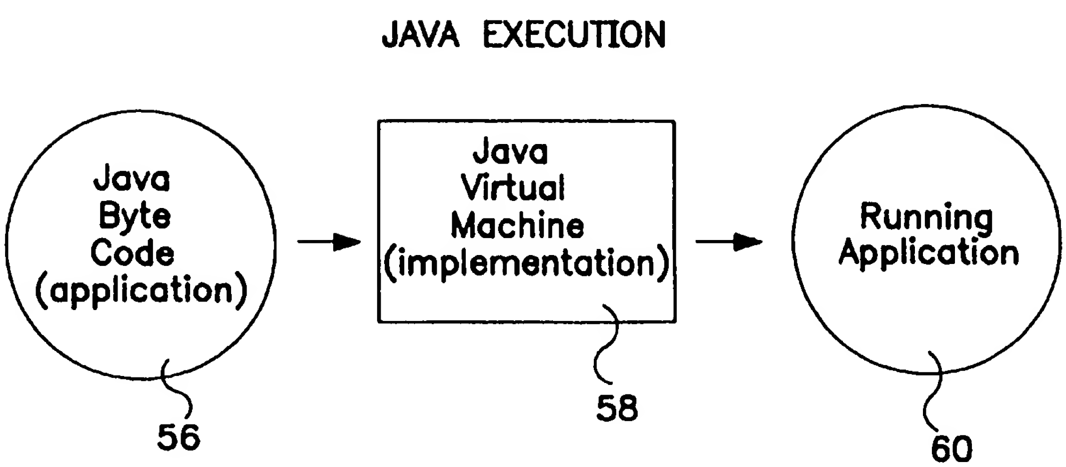


FIG. 3B
(PRIOR ART)

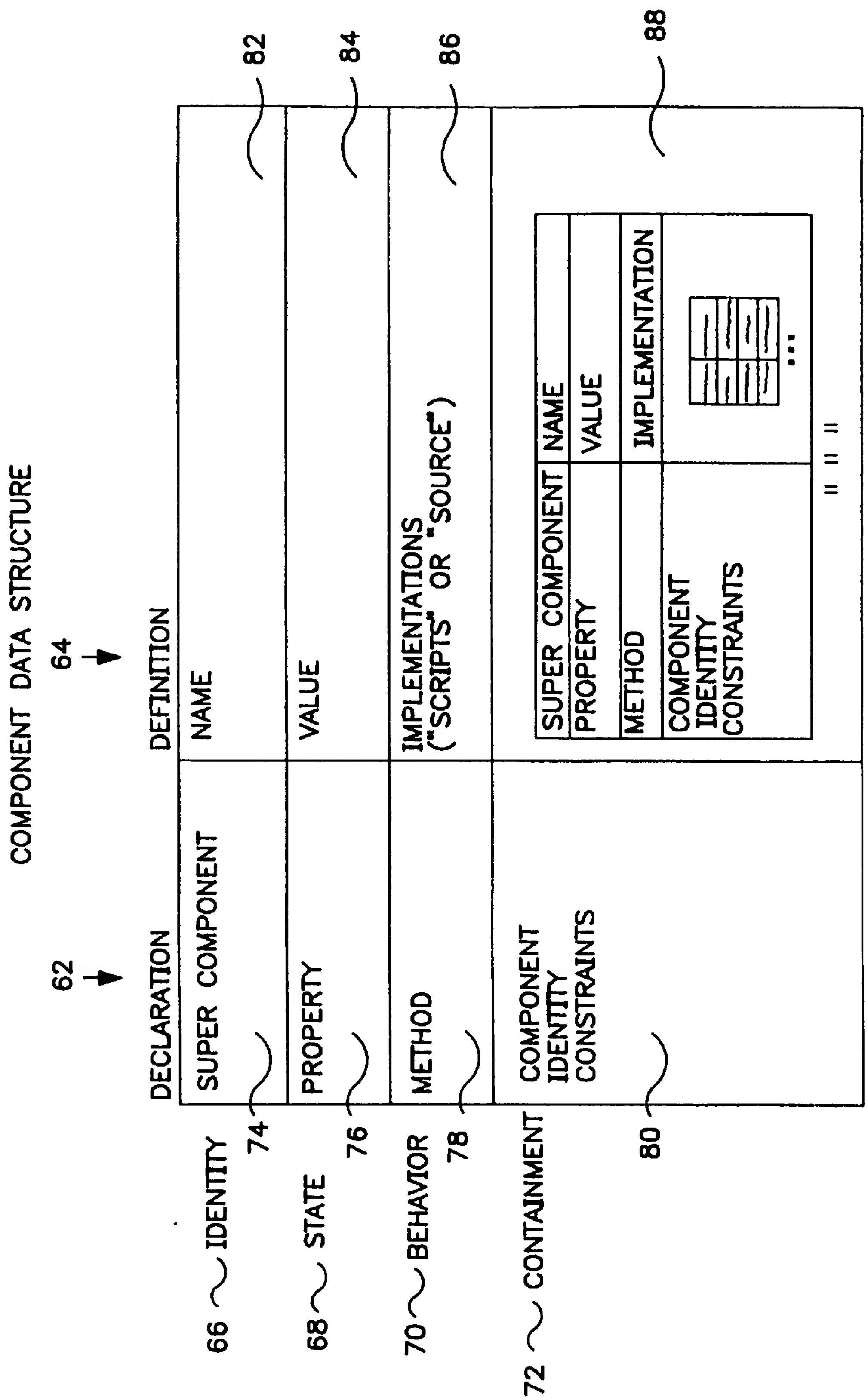
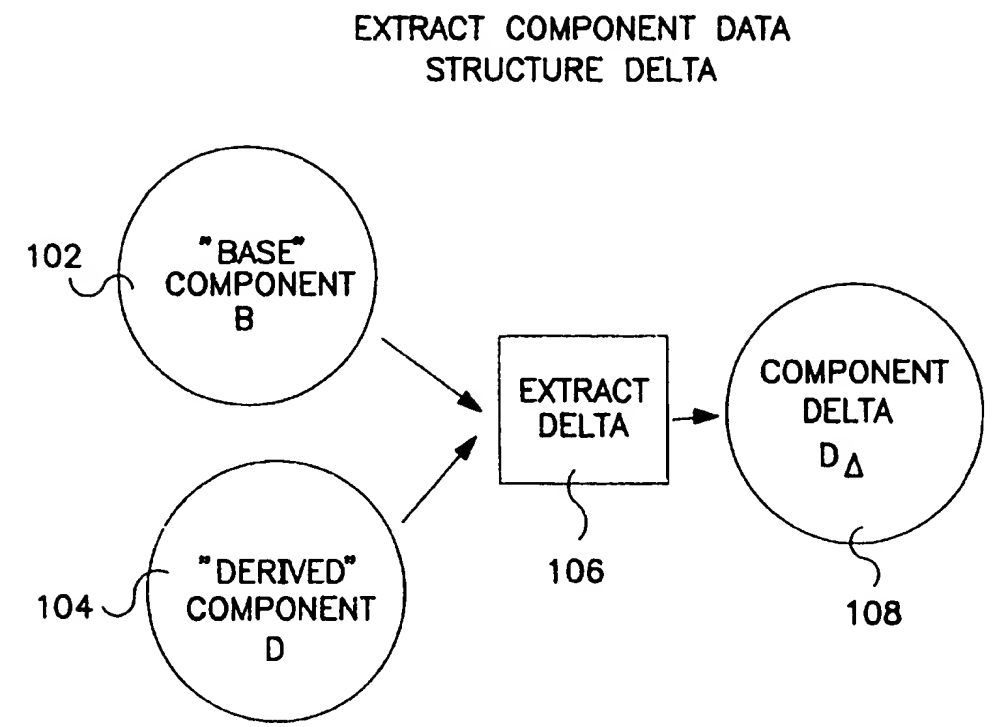
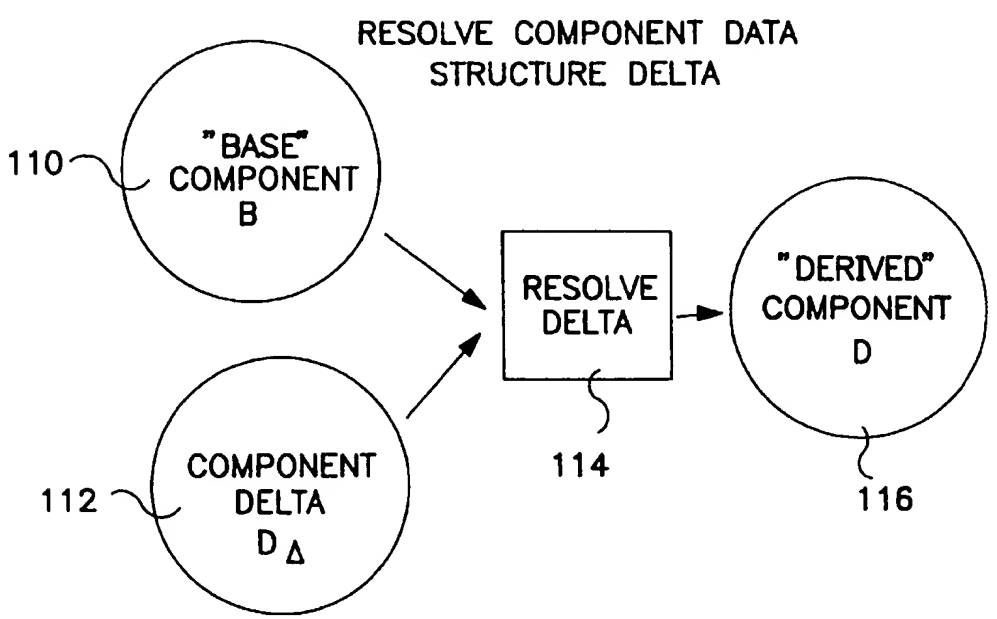

COMPONENT DATA STRUCTURE
62
64
DECLARATION
DEFINITION
66 IDENTITY
68 STATE
70 BEHAVIOR
72 CONTAINMENT
SUPER COMPONENT
NAME
PROPERTY
VALUE
METHOD
IMPLEMENTATIONS ("SCRIPTS" OR "SOURCE")
COMPONENT IDENTITY CONSTRAINTS
SUPER COMPONENT
NAME
PROPERTY
VALUE
METHOD
IMPLEMENTATION
COMPONENT IDENTITY CONSTRAINTS
74
76
78
80
82
84
86
88

FIG. 4



EXTRACT COMPONENT DATA
STRUCTURE DELTA
"BASE" COMPONENT B
102
EXTRACT DELTA
COMPONENT DELTA D_Δ
"DERIVED" COMPONENT D
104
106
108

FIG. 5A



RESOLVE COMPONENT DATA
STRUCTURE DELTA
"BASE" COMPONENT B
110
RESOLVE DELTA
"DERIVED" COMPONENT D
COMPONENT DELTA D_Δ
112
114
116

FIG. 5B

COMPILATION OF COMPONENT DATA STRUCTURE TO JAVA BYTE CODE

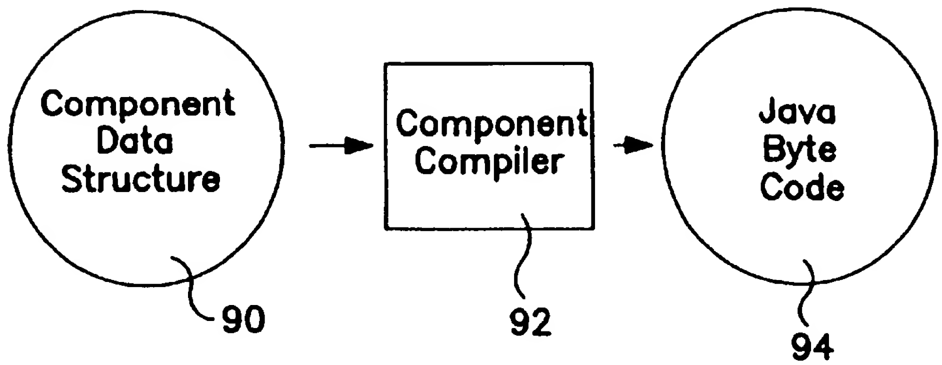


FIG. 6A

ALTERNATE COMPILATION OF COMPONENT DATA STRUCTURE TO JAVA SOURCE CODE

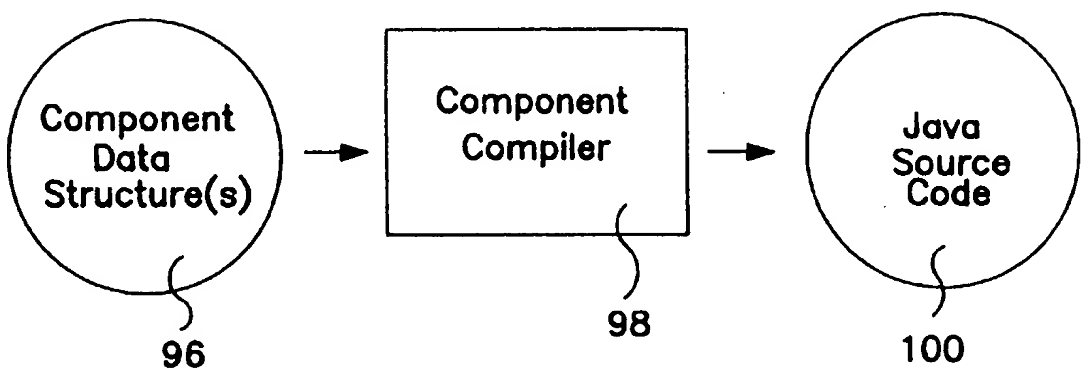


FIG. 6B

JAVA COMPATIBLE OBJECT ORIENTED COMPONENT DATA STRUCTURE

FIELD OF THE INVENTION

This invention relates generally to data structures, and more particularly to data structures that are fully compatible with the Java Virtual Machine. Even more particularly, the present invention relates to tools, incorporating these data structures, for designing software components, and constructing software applications.

BACKGROUND OF THE INVENTION

The Java language is well known in the art, and a brief review suffices for the purposes of the present invention. Java is an effective object-oriented programming language that shares many syntactical characteristics of the C and C++ languages. The popularity of the language emanates from a number of features, two of the principal (and related) ones are first that the language is portable, and, second, it is architecturally neutral. For example, some of the standardized features that make Java portable are that:

Java provides the same Application Programmer Interface (API) for all systems;

Java uses the 16-bit Unicode character set that can be used to encode all the symbols of virtually all the world's languages; and

Java supports numerical and operand evaluation standards found in most other programming languages.

Architectural neutrality is ensured by standardization on the Java Virtual Machine (JVM). This is a critical and very attractive feature of Java. When Java source code is compiled the usual output is an intermediate code called Java Byte Code UBC); this code is not directly executable. Since JBC is a platform independent standard, there is a need for a platform-specific interpreter or compiler to provide the binary code that actually executes on a specific machine.

The above features of the JVM emanate from its standardized formation of object classes that are platform independent and, via the platform specific interpreter, made compatible and operational with the particular systems. Two standards for such object classes have emerged as the most popular—the JavaBean and CORBA (Common Object Request Broker Architecture) objects. JavaBeans often are graphical controls used in (but not limited to) the construction of a Graphical User Interface (GUI). CORBA objects are capable of bridging language, machine, environment and other such boundaries. Java is described in "The Java Language Specification," Sun Microsystems Inc., JavaBeans are described in the "Java Beans API specification," version 1.00, Sun Microsystems Inc., and CORBA is described in the "Object Management Architecture Guide," 3rd edition, from the Object Management Group, Inc., all of which are incorporated herein by reference.

The widespread adoption of Java and Java compatible software and systems has made it desirable to provide software tools for applications developers that are compatible with the installed systems. The development, deployment and maintenance of large and complex software applications, such as line-of-business and packaged applications, introduce particular problems that are not typically encountered in small-scale projects. Large projects become increasingly difficult, time consuming and error prone, and inefficiencies and limitations of the technologies and derived software tools become more pronounced and noticeable. Consider the following examples:

1. A relatively minor programming change in a product, such as changing the data type returned from a Java method, requires every derived class which implements the method and every other class which invokes the modified method to be manually updated to reflect the change. Other changes are potentially much more destructive; for example, changing the name of the method may cause a conflict if a method further down the inheritance tree has the same name.
2. It is routine for large software applications to be customized (e.g. for specific customers or sites). In prior art systems, customization renders large software applications nearly impossible to maintain. For example, consider a base software application that has been installed for many different customers. Each of these customers will usually change parts of the base application to accommodate their specific needs. The changes are accomplished by making modifications to parts of the program, or in object oriented programming, by modifying the classes of objects in the application. If the application vendor subsequently modifies these classes, and the customer wishes to upgrade the application to incorporate those modifications, then the customer must determine what customizations were made to the original classes and similarly customize the updated classes. (Alternatively, if the vendor's modifications are minor, it may be possible to isolate those changes and apply them to the older but customized version of the classes.) In a large customized application, this work is substantial enough that many customers continue to use out-of-date versions indefinitely. The effort that is required to upgrade a customized application is often larger than the original customization effort. Compounding the cost of upgrading is the potential for introducing new problems into a software application that is already in use.
3. Localization is an additional requirement for many software applications and involves tailoring a software application to the language, input methods, and standards of a specific locale. This may include supporting different units of measurement and financial currencies, fulfilling legal requirements, and displaying dates, times, and currencies in the expected formats. These changes to an application provide challenges similar to those resulting from application customization; however, localization multiplies, rather than simply adding to, the number of potential application configurations, since a localized application can be customized or vice versa.

With respect to this invention, the following four implemented concepts are important. The four are: Identity, behavior, state and containment which are defined and discussed below and are well known terms of art:

Identity is used to exactly specify (to name) a class of objects, for example "button;"

Behavior consists of the operations that a class of objects can perform, for example how a "button" reacts when pressed;

State describes the current attributes of an object, for example the text displayed on a "button;" and

Containment refers to a relationship between multiple objects where one object is contained within another, for example, the "button" could be contained within a "window."

In practice, well known in the art, the above items may typically be implemented on a computer display. A "window" is typically a framed area, and a "button" appears as a smaller framed area within the window. The appearance is that of a physical window containing a physical button, where the button can be "pressed" or activated by "clicking" on it.

Inheritance is a central feature of object-oriented programming in general and the Java language in specific. In

Java, heredity is substantially limited to the identity and behavior of a class. The Java language lacks the notion of inheritance of state and containment. The state of a Java object is typically held by class variables, which the Java language calls "fields." The value of a field can be specified only in the class that declares the field. Java programs commonly use containment of objects, but not inheritance via the containment.

Object-oriented programs built using the Java language are thus limited by the lack of support for inheritance of state and containment as illustrated below.

Most of the technical terms used herein are defined by Dr. Grady Booch in his book, "Object-Oriented Analysis and Design," 2ed, published by The Benjamin/Cummings Publishing Co., Inc. in 1994, which book is incorporated herein by reference.

Illustrating the inheritance issues with state and continuing with the "button" example, it is expected a "button" would include its display position as part of its state. This implies that the "button" class, or a class that it inherits from, declares position as part of its state. The act of positioning a "button" on a "window" defines the position portion of the state of the "button." Using this example, the limitations of prior art systems become evident:

1. If the aforementioned "button" on the "window" is itself a class (which is to say that this particular "button" on this particular "window" has its own class identity), it cannot define its position, since the position was declared in the "button" super class from which this particular "button" inherits.
2. If the aforementioned "button" on the "window" is simply an instance of a generic "button" class (as in the most common prior art systems), then inheritance does not apply, since this "button" is not in and of itself a class, and therefore does not inherit.

In the two examples above, prior art systems represent state by constructing behavior. In other words, state is implemented by producing Java language instructions that modify the object instance after it is instantiated; therefore:

1. The state must also be induced from the behavior.
2. As part of the program source code, these instructions are a source of entropy, making the program more error prone, more difficult to maintain, and less capable of change.
3. The state is not inherited.

Continuing with this example, to appreciate the problem with inheritance, consider that the resulting state of this particular "button" is the culmination of its inherited state plus the changes to that state which were made on this particular "button" itself. There are two main ways which prior art systems have implemented this:

1. This particular "button" would contain the instructions required to fully configure its state as designed. This approach fails to provide inheritance because changes to classes from which this particular "button" derives would not be reflected in the instructions that configure the state of this "button."
2. This particular "button" would first execute the state configuration instructions of its super class (the class which it inherits from) and would then configure only the portion of its state that differs from the state of its super class. This approach is limited because of its recursive nature (i.e. the super class itself may have a super class); each class must redo the state configuration of its super class that differs from its own. As an example, consider the position portion of the "button" state. If the position of the super class is different from the position of this particular "button," then the "button" is positioned by the super class and then re-positioned by this "button" class. Since more than two levels of inheritance are common, the "button" could literally jump around the screen while it is being created. Although humorous as a visual example, the results would be less funny in a financial application.

Similar problems exist with containment. Generally, in the prior art, contained objects are not themselves classes and therefore do not have a class identity, an inherited state, customized behavior, or recursively containment elements capable of these same features. Furthermore, in order to support the inheritance of state, behavior, and containment of all classes, it is essential that, when a containing class is inherited, that the contained classes (and so on) be inherited as well. Since Java does not support containment, it obviously does not provide inheritance of contained objects.

FIGS. 1A and 1B illustrate general prior art principles that are characteristic of Java inheritance. Several phrases exist that describe inheritance: if a descendant class "D" inherits from an ancestor class "C," then it is said that "D is a C," that "D extends C," that "D inherits from C," that "D derives from C," and that "C is a super class of D." FIG. 1A shows a simple class inheritance hierarchy which exists in Java: a Button 2 "is a" 4 Component 6, which "is a" 8 Object 10. Furthermore, a Button 2 "is a" 4 and 8 Object 10, but this relationship is qualified as indirect, for example "Button indirectly inherits from Object" or simply "Button is a descendent of Object." (For reference, the fully qualified Java classes shown in FIG. 1A are java.lang.Object 10, java.awt.Component 6, and java.awt.Button 2.)

FIG. 1B shows a larger class inheritance hierarchy that exists in Java, incorporating the hierarchy from FIG. 1A. In this diagram, as in FIG. 1A, Component 14 "is a" Object 12, and Button 16 "is a" Component 14. Additionally, File 24 "is a" Object 12, Checkbox 18 "is a" Component 14, Container 20 "is a" Component 14, and Window 22 "is a" Container 20. The hierarchy is not limited in depth, and at any depth there can exist any number of classes. The design of the inheritance system mandates that each fully qualified class name be unique and that all classes inherit directly or indirectly from Object 12. (For reference, the fully qualified Java classes shown in FIG. 1B are java.lang.Object 12, java.awt.Component 14, java.awt.Button 16, java.awt.Checkbox 18, java.awt.Container 20, java.awt.Window 22, and java.io.File 24.)

In contrast to the "is a" of FIGS. 1A and 1B, FIGS. 2A and 2B pertain to containment. FIG. 2A illustrates a common use of containment, which is the containment of graphical objects. (The contained graphical objects are typically referred to as "controls" or "components." The containing graphical object is often a "window," "form," "pane," "site," or just "container.") Although the Java implementations of containment in the prior art vary widely, the concept of a container/containee relationship is extensively used for constructing complex objects, such as the object in FIG. 2A. In this figure, the Message 26 contains Text 28 and OK 30 and Cancel 32 buttons.

FIG. 2B extends the example slightly from FIG. 2A by adding recursive, or "nested," containment. Again, containment is a hierarchy in which a container can contain objects that are themselves containers. In FIG. 2B, Choices 38 is contained by Message 34 and Choices 38 in turn contains Shutdown 40, Restart 42, and Logoff 44. The expected visual representation of Choices 38 and the graphical controls that it contains would be a group of items, of which one and only one can be selected.

It is an object of the present invention to provide a data structure that fully describes the identity, state, behavior, and containment information related to the design of a software component. A related object is that software components contained therein be able to utilize the same data structure (a nested instance thereof) to describe their identity, state, behavior, and containment information.

It is another object of the present invention that the data structure can support the feature of inheritance with respect to the identity, state, behavior, and containment information. A related object is that software component information contained therein is inherited when the containing software component information is inherited; in other words, the inheritance applies to the entire containment hierarchy.

It is another object of the present invention that the data structure can be implemented for and can be fully compatible with the JVM, allowing use at any and all particular installations.

It is yet another object of the present invention that the data structure contains sufficient and unambiguous information in order to create Java classes from that information. A related object is that the resulting classes be Java software components capable of being assembled into or used in software applications and tools.

Still another object of the present invention is that the data structure enables the computing system to do the bulk of the work when updates or other changes are made to an instance of the data structure. Most significantly, this applies to the feature of inheritance, in which a change to the design of a software component impacts those software components that inherit from it. Additionally, this applies to software customization and localization, which produce component modifications for particular customers and locales. A related object is to enable the computing system to automatically detect and fix conflicts and redundancies caused by such changes.

Yet another object of the present invention is to provide a component data structure derived from a component data structure, wherein the derived component data structure elements can be reconstructed even if the base component data structure elements have been altered or removed. A related object of the present invention is to provide a "tag" associating the elements of a derived component data structure to the corresponding elements in the base component data structure.

Still another object of the present invention is to provide a delta component data structure wherein the elements of the delta component data structure are the differences between the elements of the base and the elements of the derived component data structures.

SUMMARY OF THE INVENTION

The above objects are met in computer systems operating in an object-oriented programming environment where the system or apparatus generates and maintains data structures for the design of components that are compatible with the Java Virtual Machine.

The data structures provide means for defining the Identity, State, Behavior and Containment of the components. The identity provides the super component from which the component inherits, and the name of the component itself. The state provides the declared properties of the component and any values for those properties. The behavior provides the declared methods of the component and any implementations for those methods. The containment declares the types of components that can be contained and includes the contained components themselves; the contained components are defined by nested versions of the inventive data structure.

The data structure finds particular advantage for providing software tools for Java compatible systems, and especially for application developers, systems integrators, and information services providers. Moreover, tools for installing and controlling revisions and modifications are more easily and accurately accommodated with the present inventive data structures since there is a reduction in the manual operations to effect changes.

Another advantage of the present invention, in a preferred embodiment, is that the data structure for one component that is derived from another may be held as the set of differences, a "delta," between the two data structures, rather than as a complete data structure. In this manner, inheritance is facilitated since the complete data structure for the derived component can be constructed by adding the delta data structure of the derived component to the complete data structure of the super component. In order to provide a robust implementation of the delta data structure, a "tag" is provided for each discrete unit of information within the component data structure; when changes are made to the super component, this tag correlates the information in the delta with the information in the super component. An additional advantage of the delta data structure is that it enables the customization and localization of components: by storing component customization and localization information as delta data structures, changes to non-customized and non-localized component data structures are inherited by their customized and localized counterparts. A further advantage of the delta data structure is that it uses less memory and storage space.

The inventive component data structures find further advantage by providing means for automatically, via the computer systems, incorporating changes among upgraded, customized and localized software applications, particularly in widely dispersed installations incorporating large software systems. Since changes made manually to a single component data structure are then automatically propagated to the impacted component data structures, the potential for human error is markedly reduced.

Other objects, features and advantages will be apparent from the following detailed description of preferred embodiments thereof taken in conjunction with the accompanying drawings in which:

BRIEF DESCRIPTION OF DRAWINGS

FIGS. **1A** and **1B** are hierarchical diagrams of prior art inheritance;

FIGS. **2A** and **2B** are hierarchical diagrams of prior art containment;

FIG. **3** is a diagram of a simplified prior art Java system;

FIG. **4** is a diagram of the innovative data structure;

FIGS. **5A** and **5B** are diagrams of how component inheritance is managed; and

FIGS. **6A** and **6B** are diagrams of compiling the data structure for the JVM

DETAILED DESCRIPTION OF PREFERRED EMBODIMENTS

FIGS. **3A** and **3B** present a simplified typical Java system. In FIG. **3A**, the Java compiler **52** transforms the input, Java source code **50**, into a structure known as Java Byte Code **54** (also referred to as the "Java.class file format"). This Java Byte Code **54** exists as an interim step, which is then

provided as input 56 in FIG. 3B to an implementation of the Java Virtual Machine 58, resulting in a running application 60. It should be noted that in some installations, the raw Java source code could be compiled to a platform-dependent executable code structure, omitting the intermediate step of Java Byte Code. However, most applications follow the system as in FIGS. 2A and 2B, and those that compile directly to executable code must still conform to the Java Virtual Machine Specification, (Sun Microsystems Inc., copyright 1997).

The import of FIGS. 3A and 3B is the realization that the JVM is universal and platform independent. This realization provides a path where software components, applications, and tools can be provided that are also universal and platform independent. In order to achieve this, there exists the single requirement of compatibility with the JVM standard. Once Java Byte Code 56 is formed, all of the many computing platforms that provide a JVM implementation 58 will be able to run the code 60. Furthermore, according to the Java Virtual Machine Specification, the JVM is independent of the source from which the Java Byte Code is formed: "the Java Virtual Machine does not assume that the instructions it executes were generated from Java source code. [There] have been a number of efforts aimed at compiling other languages to the Java Virtual Machine . . . " (Java Virtual Machine Specification, Chapter 7, "Compiling for the Java Virtual Machine").

Components are object classes in the broadest sense. The term component refers both to the design of the object class and the resulting output of the design. In the present invention, a preferred embodiment of a component is a data structure composed of identity, state, behavior, and containment, as displayed in FIG. 4. A preferred embodiment of the resulting compiled output is Java Byte Code, as illustrated in FIG. 6A. Additionally, the Java Byte Code adheres to the JavaBean standard. For these purposes, the term "component" will be used hereinafter.

A preferred embodiment of the innovative component data structure is shown in FIG. 4. The organization of the data structure reflects the composition of a component as previously described: identity 66, state 68, behavior 70, and containment 72. Additionally, the data structure is conceptually divided between declaration 62 and definition 64; declaration describes information about a trait of the component and definition provides the actual "value."

In FIG. 4, identity 66 declares the "super component" 74 which is inherited from, and the identity or name 82 of the inheriting component. There exists exactly one "root component," named "Component" that has no super component; all other components inherit directly or indirectly from the root component. (A single root in an inheritance hierarchy is well known in the art. For example, Java has a root class known as java.lang.Object.) A preferred embodiment is such that the unique identity of a given component is constructed from both its super component 74 and its name 82. This provides hierarchical organization by inheritance. Furthermore, all components inheriting from a given component share a namespace unique to the component from which they inherit. For example, a component with the name 82 "Control" and inheriting from the root component ("Component") would have a unique identity "Component.Control." (The hierarchical delimitation using a period is a standard in Java.) There are further elements in the declaration of a component's identity that reflect the particular standards and systems with which the component must comply. For example, the JavaBean specification defines how a component "dispatches" events to a JavaBean event listener interface.

In FIG. 4, state 68 declares properties 76 for which each may have a value 84 defined. The declaration of a property 76 most notably includes a name of the property, which uniquely identifies it within the component, and a data type. There are further elements in the declaration of a property that reflect the particular standards and systems with which the component must comply. For example, the JavaBean specification supports multiple values for an "indexed" property. The value 84 of a property is constrained by the declaration of the property 76, for example by the data type.

In FIG. 4, behavior 70 declares methods 78 each with optional implementations 86 defined. The declaration of a method 78 most notably includes the name and parameters of the behavior, which uniquely identifies it within the component, and a return data type. There are further elements in the declaration of a behavior that reflect the particular standards and systems with which the component must comply. For example, the Java Language Specification provides a manner to declare "exceptions" that a method can "throw." Each implementation 86 provides the information necessary to be compiled into the instruction code of the particular systems. A preferred embodiment specifies the "language" of implementation and an implementing "script," where the language is used to determine the manner of compiling the script.

In FIG. 4, containment 72 declares component identity constraints 80, which are the types of components that can be contained within this component, and defines those components 88 that are contained. The ability to declare containment 72 is based on the unique identity 66 that each component has. For example, if this component identity constraint 80 limits containment to "Component.Control," then only the component "Component.Control" and any component inheriting directly or indirectly from it may be contained within this component.

In FIG. 4, this same component data structure is used recursively to hold the component information for a contained component 88. The only difference is that the contained component's unique identity is constructed from its name and its containing component. For example, if a component inheriting from "Component.Control.Button" and named "OK" is contained within the component "Component.Control.Container.Window.Message" (see FIG. 2A), then the unique identity of the OK button is "Component.Control.Container.Window.Message$OK." In other words, the containing component provides a namespace for the components it contains. (The delimitation using a dollar sign is a standard in Java for inner classes.)

In order to support component inheritance, features such as containment that rely on inheritance, and the customization and localization of components, it is necessary that the differences between two components be both determinable and applicable. For these purposes, the terms Base, Derived, and Delta will be defined:

Base—A data structure assumed to contain complete and correct information;

Derived—A data structure originally identical to the Base data structure but which may have since changed; and

Delta—The specific changes that must be applied to the Base data structure in order to produce the Derived data structure.

FIGS. 5A and 5B illustrate the above terms with respect to the component data structure. In FIG. 5A, a component delta "D_Δ" 108 is extracted 106 from the derived component "D" 104 of the base component "B" 102. Thus, it must be possible to determine those differences between "B" and "D" that are legal and store those differences in the form of a delta.

By keeping the resulting component delta "D_Δ" **108**, subsequent changes to the base component "B" are reflected in the derived component "D" without losing those qualities of "D" that made it different from "B." This is illustrated in FIG. **5B**, in which the application ("resolve delta" **114**) of the component delta "D_Δ" **112** to the base component "B" **110** results in the derived component "D" **116**.

In a preferred embodiment, there are two categories of differences ("deltas") between components:

1. A component derivation represents the differences between a component and its super component; the component derivation is therefore the result of an "inheritance" relationship between two components
2. A component modification represents the changes made to a component as a result of versioning, customization, or localization; the component modifications is therefore the result of comparing two different "versions" of the same component

In a preferred embodiment, both the component derivation and modification are themselves component data structures; however, instead of carrying complete component information, these data structures carry only the information necessary to reconstruct the derived component from the base component. Additionally, each element (for example, property, method, and contained component) in a base component and the corresponding element in a derived and/or modified component includes a uniquely identifying datum (a "tag", which is an implementation of a unique identifies, UID, a term well known in the art) which is used to ensure the integrity and organizations for a correct reconstruction of the derived component even if elements of the base component have been added, removed, and renamed. When the delta elements (see above) are originally extracted, the tag for the elements of the base component is duplicated in the corresponding elements of the derived component thereby uniquely associating the base and derived elements to each other. The tag for each particular element remains constant for the life of that element.

FIG. **6A** illustrates the manner in which the innovative data structure is compiled into Java Byte Code, thus fulfilling the single requirement of compatibility with the JVM standard illustrated in FIG. **3B**. It is therefore important to ensure that the component data structure is constrained by the JVM specification such that legal Java Byte Code will be produced for the component data structure. As shown in FIG. **6B**, it is also possible that the component data structure be compiled first into Java source code and then, as shown in FIG. **3A**, be compiled into Java Byte Code.

In a preferred embodiment, the component data structure is written in Java, is compatible with the JVM, and compiles to produce legal JVM classes. The component data structure must be such that it can be stored and, when stored, it can be loaded, that the computing systems accommodate the component, and that the component meets any applicable standard.

It will now be apparent to those skilled in the art that other embodiments, improvements, details and uses can be made consistent with the letter and spirit of the foregoing disclosure and within the scope of this patent, which is limited only by the following claims, construed in accordance with the patent law, including the doctrine of equivalents.

What is claimed is:

1. An object-oriented component data structure of a Java compatible type for defining components, for use in a computer system providing enhance inheritance, comprising:

a) means for defining a first component data structure and a second component data structure,

b) means for associating said first with said second component data structures, thereby allowing said first component data structure to inherit from said second component data structure,

c) means for organizing said first and second component data structures in a hierarchy according to said inheritance,

d) means for querying and modifying said first and second component data structures,

e) means for maintaining the integrity and organization of said first and second component data structures

f) means for containing said component data structures and the inheriting of said contained component data structures to be recursive,

g) means for defining the identity, state, and behavior of contained component data structures, and

h) means for the computer system to organize the contained component data structures in a hierarchy according to containment.

2. The object oriented component data structure as defined in claim **1** further comprising means for defining multiple hierarchical levels according to inheritance of data structures.

3. The object oriented component data structure as defined in claim **1** wherein the first component data structure comprises means for defining an identity of said first component data structure, wherein said identity inherits from said second component data structure.

4. The object oriented component data structure as defined in claim **3** further comprising:

a) means for defining a state of said first component data structure, and

b) means for defining a behavior of said first component data structure,

wherein said state and behavior of said first component data structure are inherited from said second component data structure.

5. The object oriented component data structure as defined in claims **4** further comprising:

a) means for containing third component data structures within the second component data structure, and

b) means for containing fourth component data structures within the first component data structure, wherein said fourth component data structures inherit from the third component data structures.

6. The object oriented component data structure as defined in claim **5** wherein identity comprises the name of the component data structure, wherein said state comprises the properties of the component data structure and the value of said properties, wherein said behavior comprises how the component data structure operates and the code necessary to implement said operation, and wherein containment comprises the component data structures that are contained within the component data structure.

7. The object oriented component data structure as defined in claim **6** wherein said data structure is substantially consistent with the Java Virtual Machine standard, and further comprising means for creating Java Virtual Machine class file structures.

8. The object oriented component data structure as defined in claim **6** wherein said data structure is substantially consistent with the JavaBean component model standard, and further comprising means for creating Java Virtual Machine class file structures which are JavaBean components.

9. The object oriented component data structure for components as defined in claim 5 wherein said first data structure comprises:

f) means for uniquely tagging each element in said second data structure,

g) means for tagging each element in said first data structure, wherein said tagging relate the corresponding elements in the first and the second data structures, so that changes to elements in the second data structure can be related to the corresponding elements in the first data structure.

10. The object oriented component data structure as defined in claim 5 wherein said first component data structure comprises:

means for determining the differences between said first and said second component data structure, and

means for storing as a delta the information that is different between said first and said second component data structures.

11. A method for defining object-oriented component data structures of Java compatible types for defining components, for use in a computer system, providing enhanced inheritance, comprising the steps of:

a) defining a first component data structure and a second component data structure,

b) associating said first with said second component data structures, thereby allowing said first component data structure to inherit from said second component data structure,

c) organizing said first and second component data structures in a hierarchy according to said inheritance,

d) querying and modifying said first and second component data structures,

e) maintaining and retaining the integrity and organization of said first and second component data structures in said hierarchy,

f) containing component data structures and the inheriting of said contained component data structures to be recursive,

g) defining the identity, state, and behavior of contained component data structures, and

h) organizing the contained component data structures in a hierarchy according to containment using a computer system.

12. The method as defined in claim 11 further comprising the step of defining multiple hierarchical levels according to inheritance of data structures.

13. The method as defined in claim 12 wherein the step of defining a first component data structure comprises the step of:

identifying said first component data structure, wherein

said identity inherits from said second component data structure.

14. The method as defined in claim 13 further comprising the steps of:

defining a state of said first component data structure, and

defining a behavior of said first component data structure,

wherein said state and behavior of said first component data structure are inherited from said second component data structure.

15. The method as defined in claims 14 further comprising the steps of:

a) containing third component data structures within the second component data structure, and

b) containing fourth component data structures within the first component data structure, wherein said fourth component data structures inherit from the third component data structures.

* * * * *

US005889941A

United States Patent [19]

Tushie et al.

[11] **Patent Number: 5,889,941**

[45] **Date of Patent: Mar. 30, 1999**

[54] **SYSTEM AND APPARATUS FOR SMART CARD PERSONALIZATION**

[75] Inventors: **David R. Tushie**, Eden Prairie; **William W. Haeuser**, Chaska, both of Minn.

[73] Assignee: **UBIQ Inc.**, Minneapolis, Minn.

[21] Appl. No.: **755,459**

[22] Filed: **Nov. 22, 1996**

Related U.S. Application Data

[60] Provisional application No. 60/015,743 Apr. 15, 1996.

[51] **Int. Cl.**^{6} **H04F 1/00**

[52] **U.S. Cl.** **395/186**; 257/679; 235/380

[58] **Field of Search** 395/186, 187.07, 395/188.01; 380/25, 23; 257/679, 922; 455/558; 902/25, 26; 235/380; 340/825.33

[56] **References Cited**

U.S. PATENT DOCUMENTS

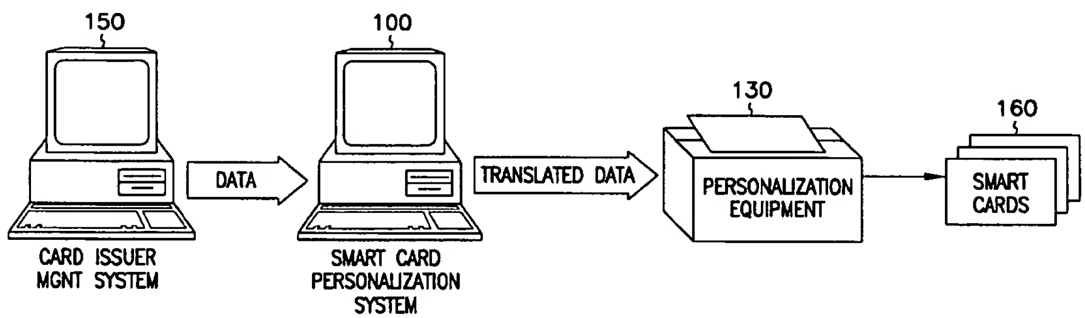
4,772,782	9/1988	Nonat	235/380
4,825,054	4/1989	Rust et al.	235/380
4,827,425	5/1989	Linden	364/478
4,866,259	9/1989	Bonnemoy .	
4,874,935	10/1989	Younger .	
4,882,474	11/1989	Anderl et al. .	
5,025,399	6/1991	Wendt et al.	364/519
5,266,781	11/1993	Warwick et al.	235/375
5,332,889	7/1994	Lundstrom et al.	235/380
5,378,884	1/1995	Lundstrom et al.	235/441
5,442,165	8/1995	Atsumi et al.	235/492
5,534,857	7/1996	Laing et al.	340/825.34
5,578,808	11/1996	Taylor	235/380
5,684,742	11/1997	Bublitz et al.	365/189.01

Primary Examiner—Albert Decady
Attorney, Agent, or Firm—Schwegman, Lundberg, Woessner & Kluth, P.A.

[57] **ABSTRACT**

A smart card personalization system maintains a database containing card issuer data format templates, card applications, card operating system commands, and personalization equipment specifications and provides a centralized interface of inputs and outputs to a card issuing process which dynamically adjusts to changes in the issuing process to easily permit a card issuer to change data formats, card applications, card operating systems and/or personalization equipment in a card issuing process. The system interfaces to any card issuer management system, manages the transfer of card holder data and card applications to the particular personalization equipment used, and maintains statistics for real-time and off-line inquiries to support critical management and reporting functions. Furthermore, the system works with a variety of security methodologies to prevent fraud.

26 Claims, 18 Drawing Sheets



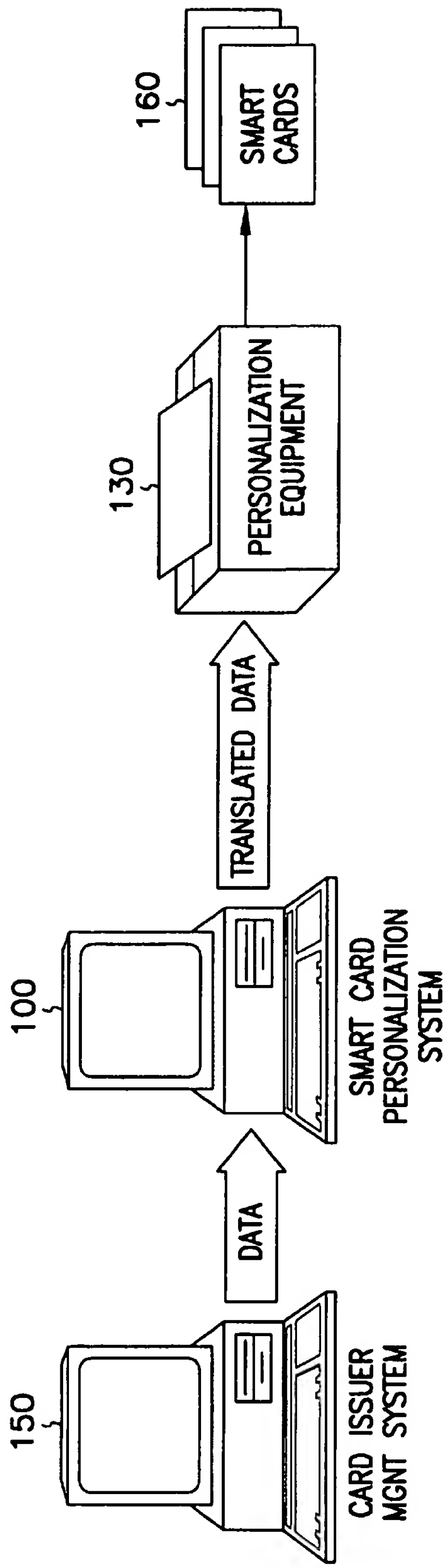

160
SMART CARDS
130
PERSONALIZATION EQUIPMENT
TRANSLATED DATA
100
SMART CARD PERSONALIZATION SYSTEM
DATA
150
CARD ISSUER MGNT SYSTEM

FIG. 1A

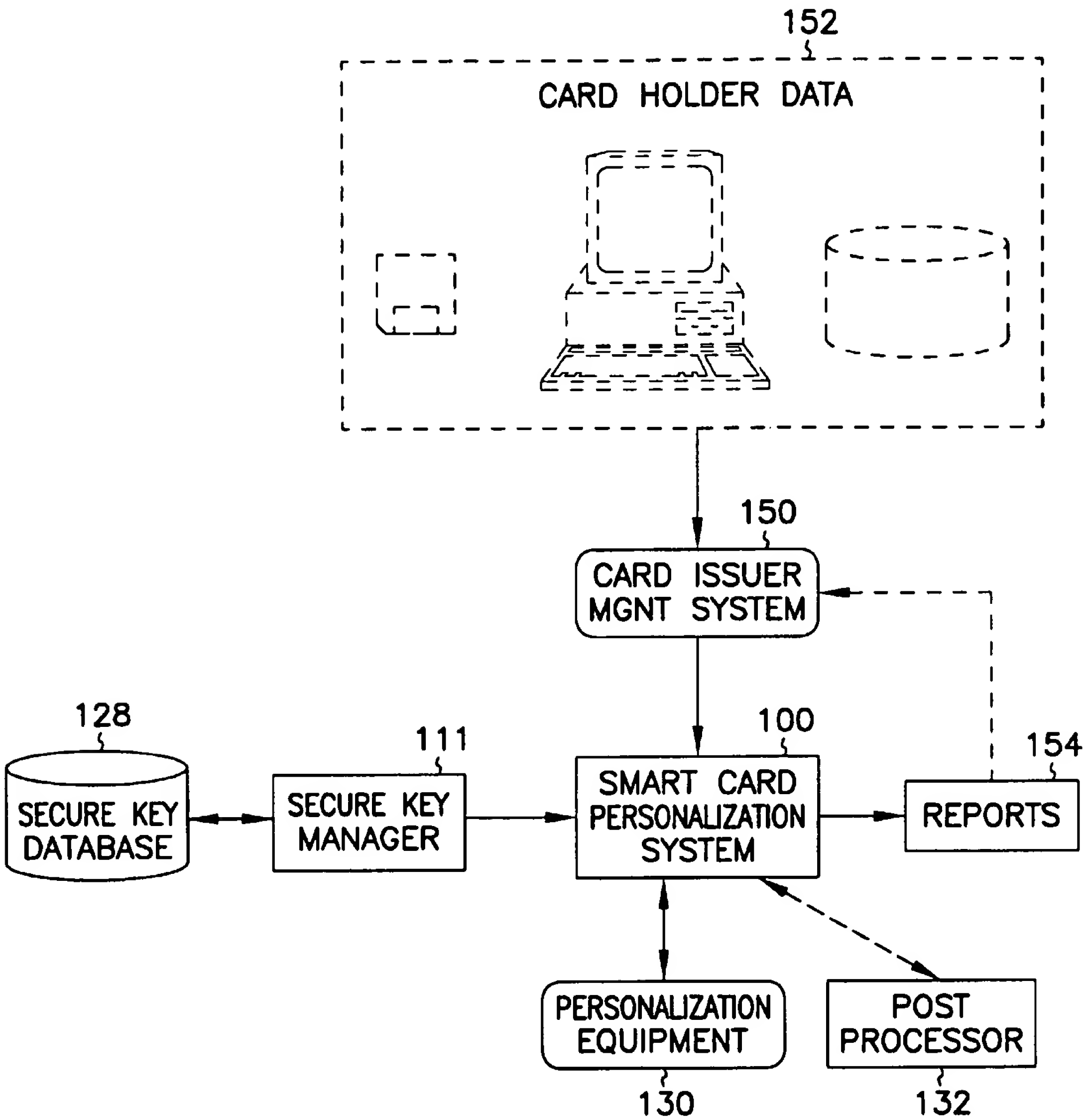

152
CARD HOLDER DATA
150
CARD ISSUER MGNT SYSTEM
128
SECURE KEY DATABASE
111
SECURE KEY MANAGER
100
SMART CARD PERSONALIZATION SYSTEM
154
REPORTS
PERSONALIZATION EQUIPMENT
130
POST PROCESSOR
132

FIG. 1B

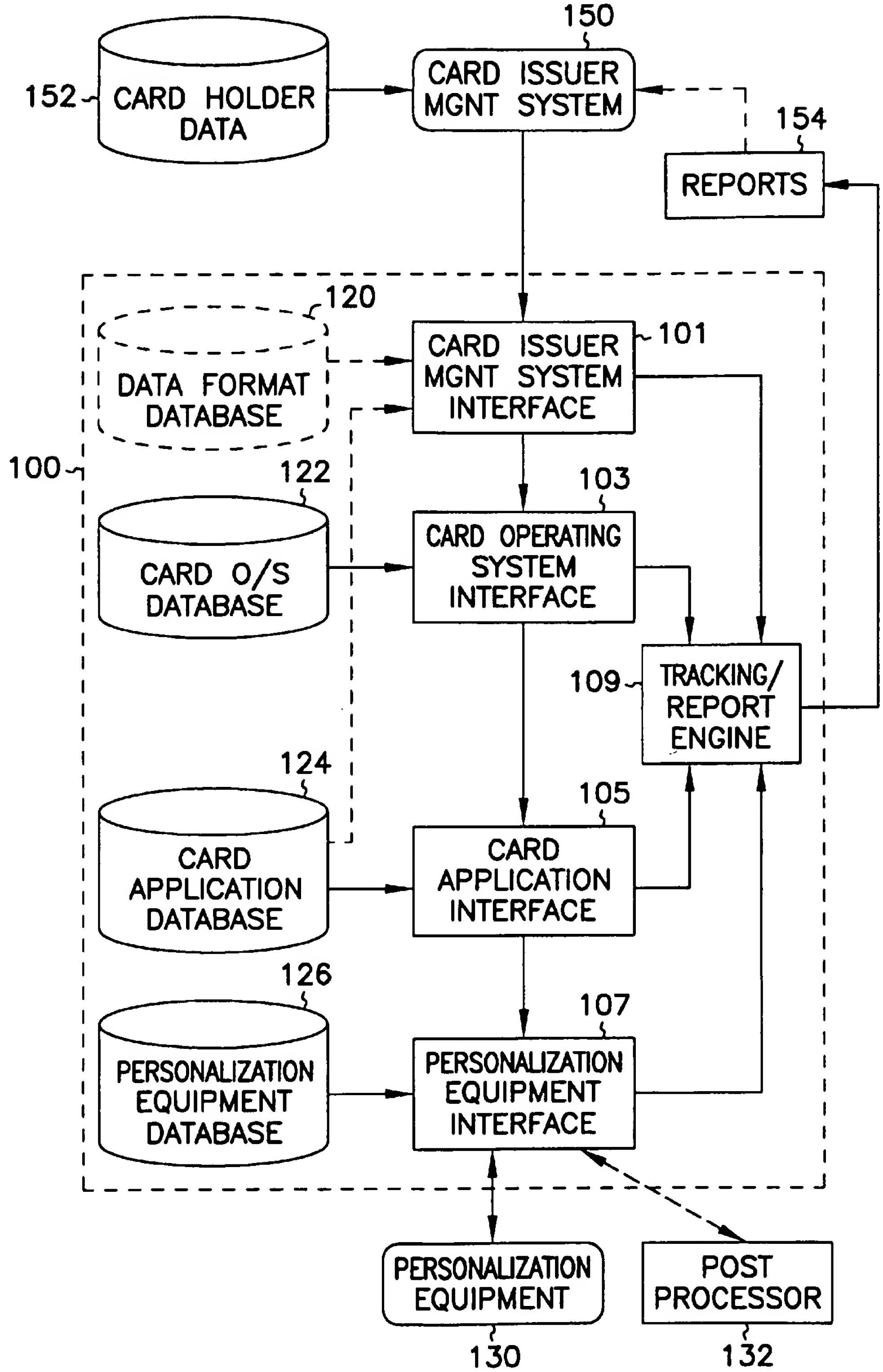

150
CARD ISSUER MGNT SYSTEM
152
CARD HOLDER DATA
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
122
CARD O/S DATABASE
103
CARD OPERATING SYSTEM INTERFACE
109
TRACKING/ REPORT ENGINE
124
CARD APPLICATION DATABASE
105
CARD APPLICATION INTERFACE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
130
POST PROCESSOR
132

FIG. 1C

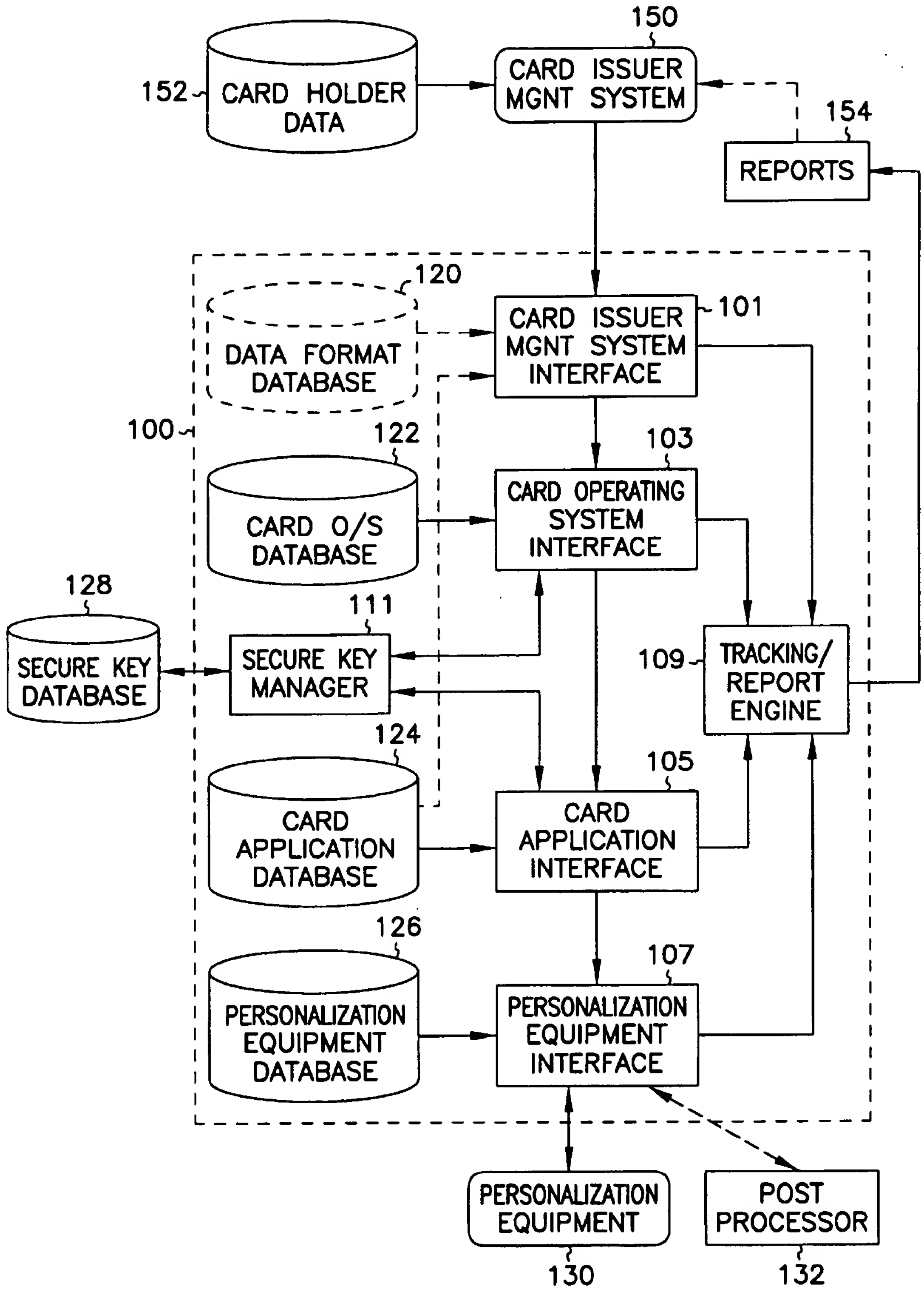

150
CARD ISSUER MGNT SYSTEM
152
CARD HOLDER DATA
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
122
CARD O/S DATABASE
103
CARD OPERATING SYSTEM INTERFACE
128
SECURE KEY DATABASE
111
SECURE KEY MANAGER
109
TRACKING/ REPORT ENGINE
124
CARD APPLICATION DATABASE
105
CARD APPLICATION INTERFACE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
130
POST PROCESSOR
132

FIG. 2

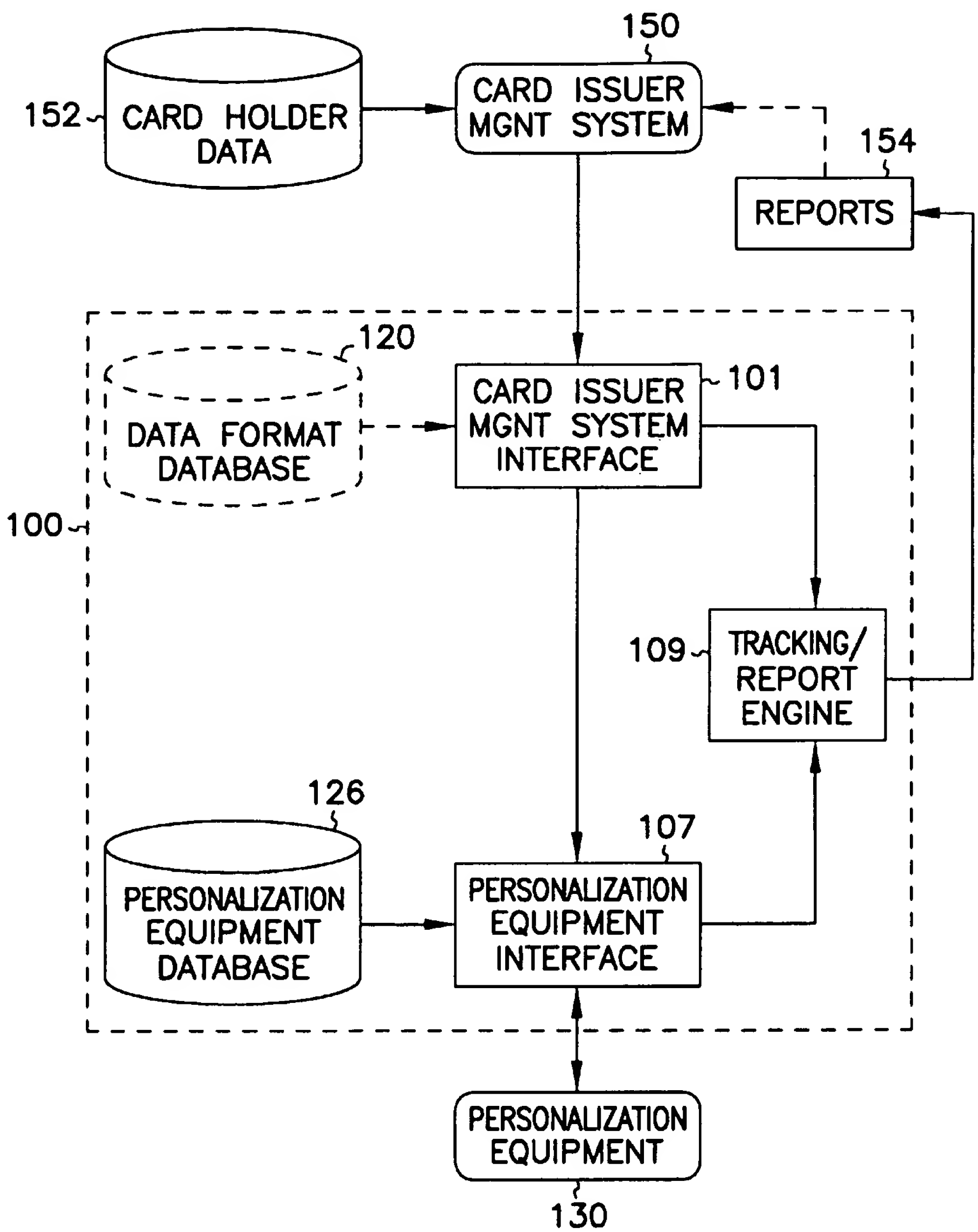

150
CARD ISSUER MGNT SYSTEM
152
CARD HOLDER DATA
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
109
TRACKING/ REPORT ENGINE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
130

FIG. 3

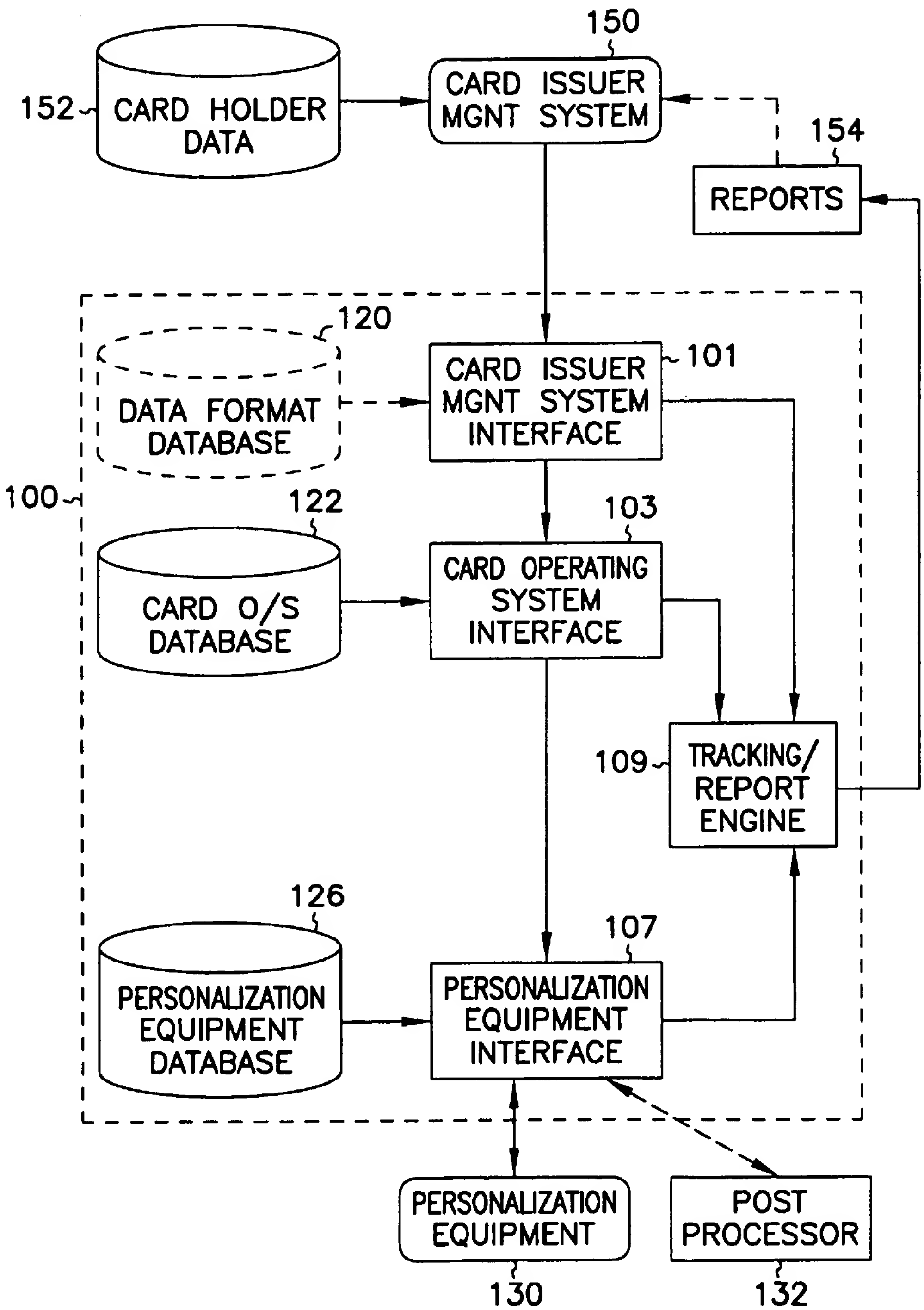
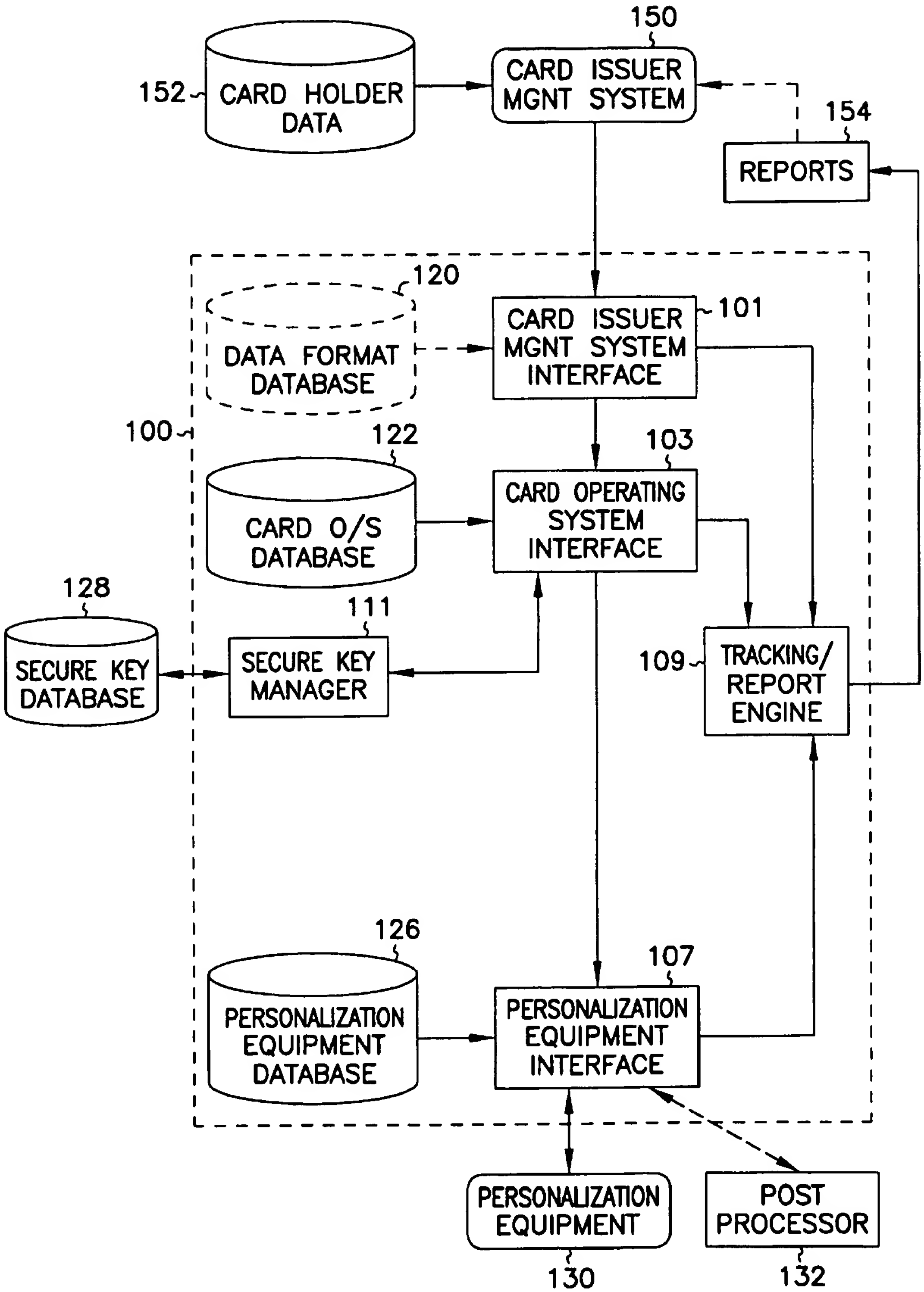

150
152
CARD HOLDER DATA
CARD ISSUER MGNT SYSTEM
154
REPORTS
120
101
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
100
122
103
CARD O/S DATABASE
CARD OPERATING SYSTEM INTERFACE
109
TRACKING/ REPORT ENGINE
126
107
PERSONALIZATION EQUIPMENT DATABASE
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
POST PROCESSOR
130
132

FIG. 4



150
CARD ISSUER MGNT SYSTEM
152
CARD HOLDER DATA
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
122
CARD O/S DATABASE
103
CARD OPERATING SYSTEM INTERFACE
128
SECURE KEY DATABASE
111
SECURE KEY MANAGER
109
TRACKING/ REPORT ENGINE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
130
POST PROCESSOR
132

FIG. 5

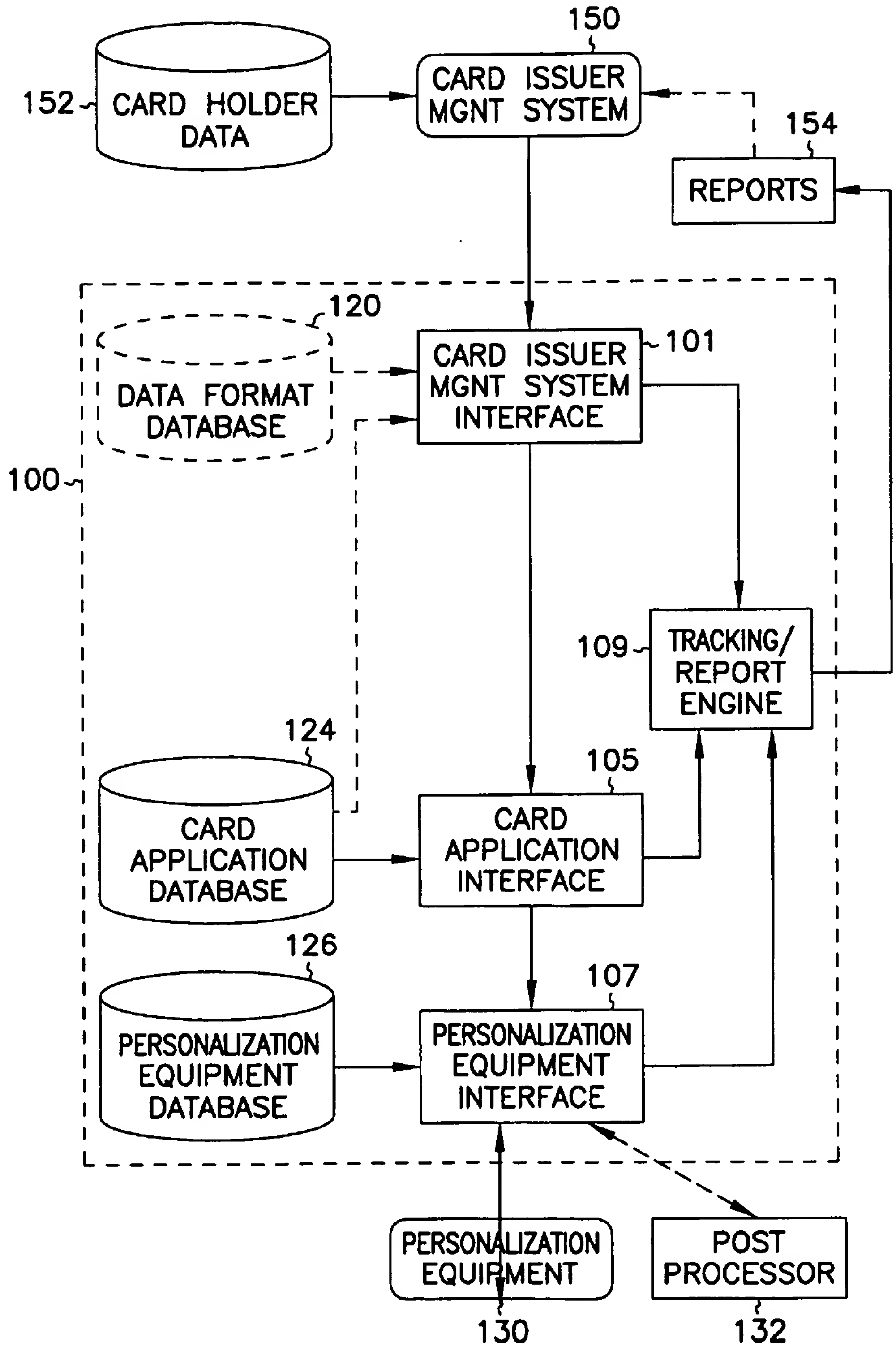

150
152
CARD HOLDER DATA
CARD ISSUER MGNT SYSTEM
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
109
TRACKING/ REPORT ENGINE
124
CARD APPLICATION DATABASE
105
CARD APPLICATION INTERFACE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
POST PROCESSOR
130
132

FIG. 6

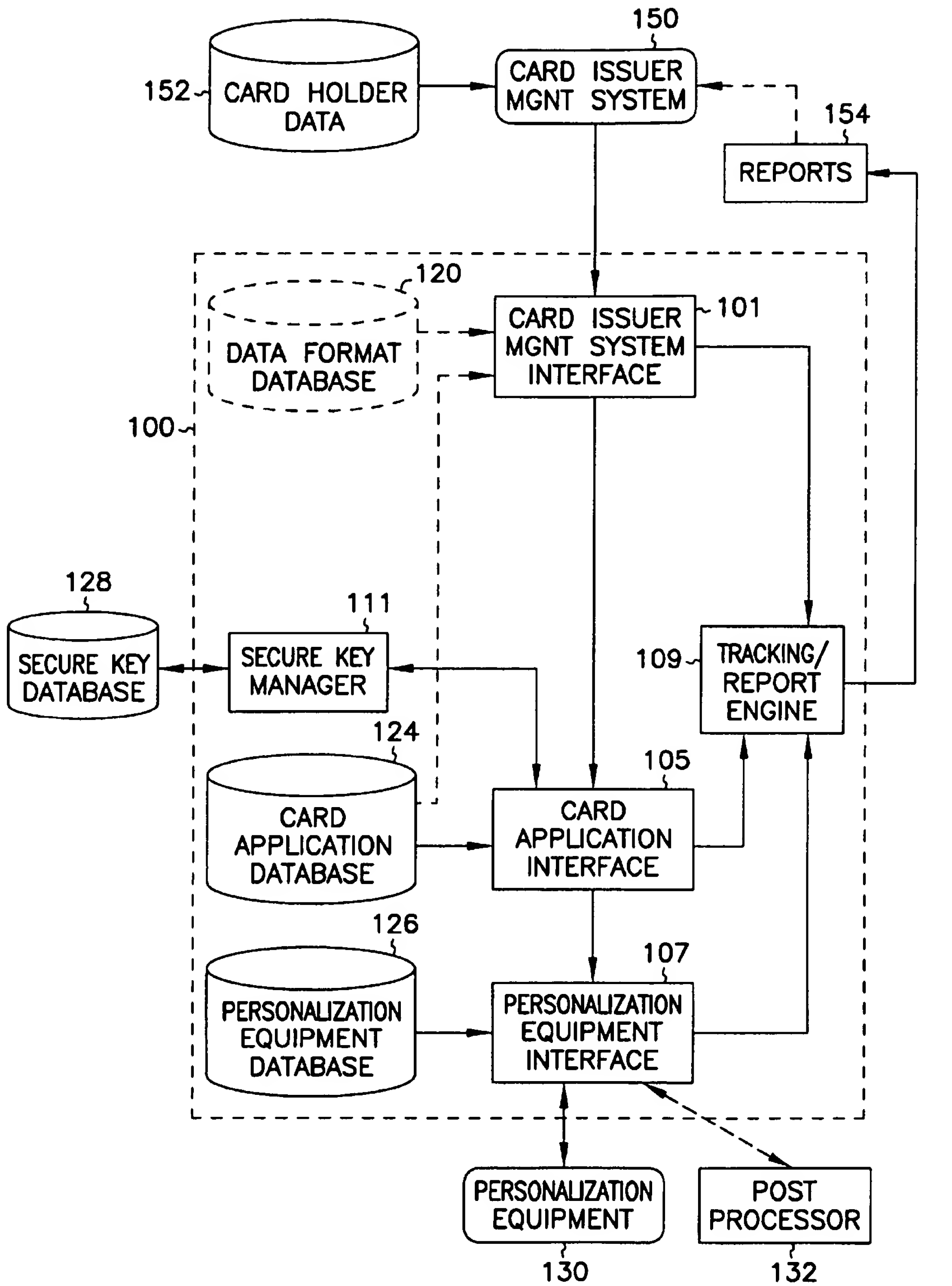

150
CARD ISSUER MGNT SYSTEM
152
CARD HOLDER DATA
154
REPORTS
120
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
101
100
128
SECURE KEY DATABASE
111
SECURE KEY MANAGER
109
TRACKING/ REPORT ENGINE
124
CARD APPLICATION DATABASE
105
CARD APPLICATION INTERFACE
126
PERSONALIZATION EQUIPMENT DATABASE
107
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
130
POST PROCESSOR
132

FIG. 7

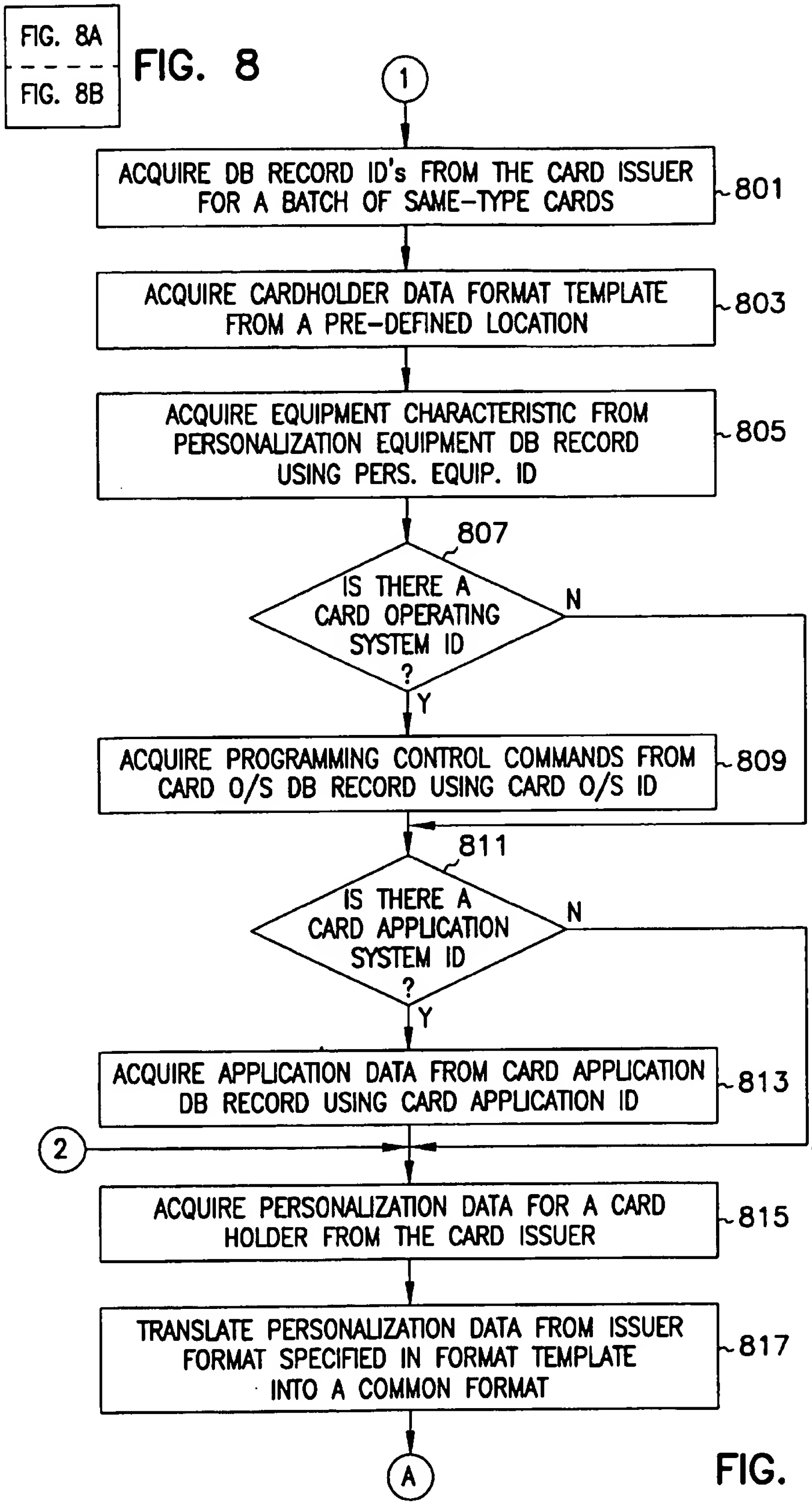


FIG. 8A
FIG. 8B
FIG. 8
1
ACQUIRE DB RECORD ID's FROM THE CARD ISSUER FOR A BATCH OF SAME-TYPE CARDS
801
ACQUIRE CARDHOLDER DATA FORMAT TEMPLATE FROM A PRE-DEFINED LOCATION
803
ACQUIRE EQUIPMENT CHARACTERISTIC FROM PERSONALIZATION EQUIPMENT DB RECORD USING PERS. EQUIP. ID
805
807
IS THERE A CARD OPERATING SYSTEM ID ?
N
Y
ACQUIRE PROGRAMMING CONTROL COMMANDS FROM CARD O/S DB RECORD USING CARD O/S ID
809
811
IS THERE A CARD APPLICATION SYSTEM ID ?
N
Y
ACQUIRE APPLICATION DATA FROM CARD APPLICATION DB RECORD USING CARD APPLICATION ID
813
2
ACQUIRE PERSONALIZATION DATA FOR A CARD HOLDER FROM THE CARD ISSUER
815
TRANSLATE PERSONALIZATION DATA FROM ISSUER FORMAT SPECIFIED IN FORMAT TEMPLATE INTO A COMMON FORMAT
817
A

FIG. 8A

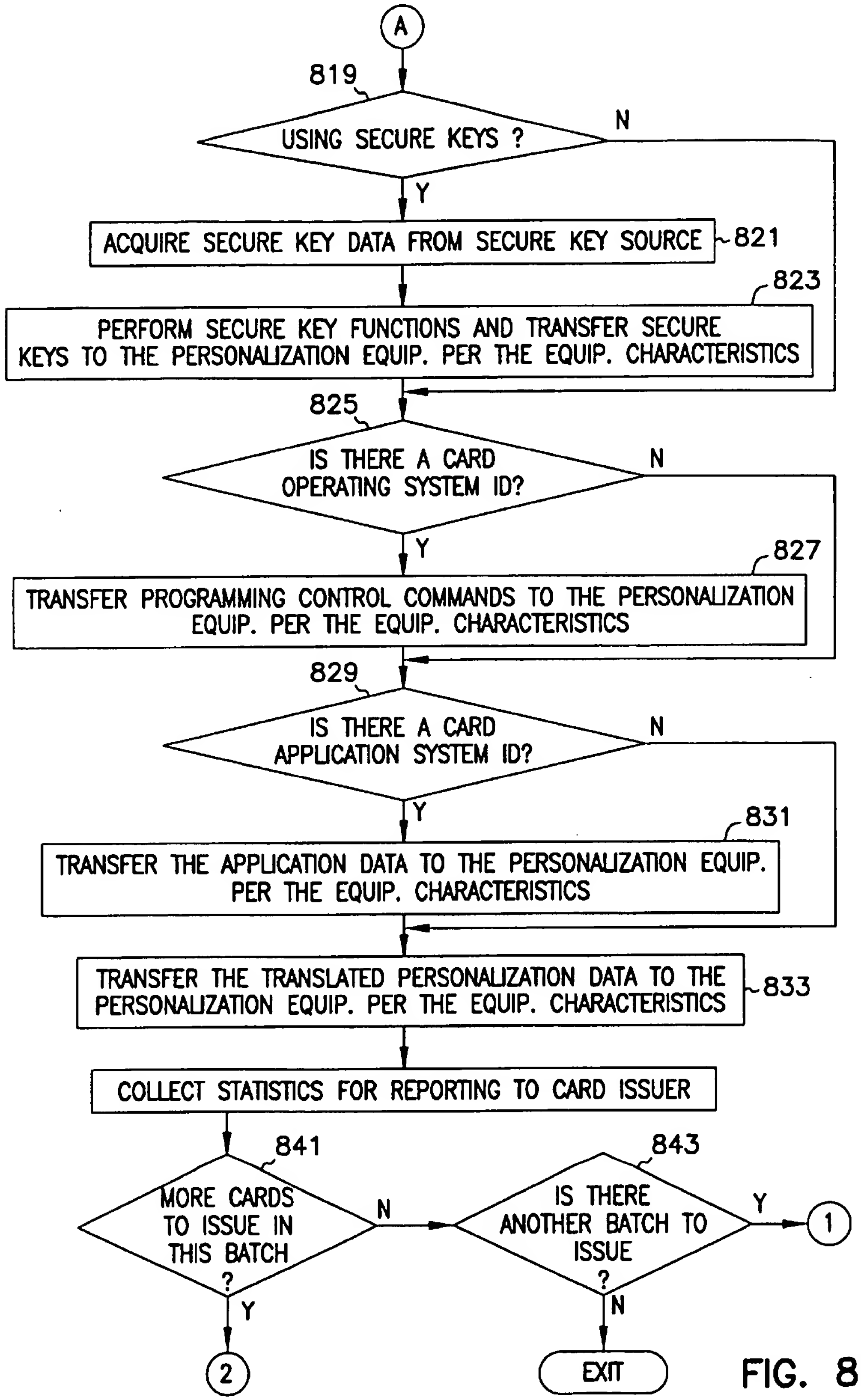

A
819
USING SECURE KEYS ?
N
Y
ACQUIRE SECURE KEY DATA FROM SECURE KEY SOURCE
821
823
PERFORM SECURE KEY FUNCTIONS AND TRANSFER SECURE KEYS TO THE PERSONALIZATION EQUIP. PER THE EQUIP. CHARACTERISTICS
825
IS THERE A CARD OPERATING SYSTEM ID?
N
Y
827
TRANSFER PROGRAMMING CONTROL COMMANDS TO THE PERSONALIZATION EQUIP. PER THE EQUIP. CHARACTERISTICS
829
IS THERE A CARD APPLICATION SYSTEM ID?
N
Y
831
TRANSFER THE APPLICATION DATA TO THE PERSONALIZATION EQUIP. PER THE EQUIP. CHARACTERISTICS
TRANSFER THE TRANSLATED PERSONALIZATION DATA TO THE PERSONALIZATION EQUIP. PER THE EQUIP. CHARACTERISTICS
833
COLLECT STATISTICS FOR REPORTING TO CARD ISSUER
841
MORE CARDS TO ISSUE IN THIS BATCH ?
N
843
IS THERE ANOTHER BATCH TO ISSUE ?
Y
1
Y
N
2
EXIT

FIG. 8B

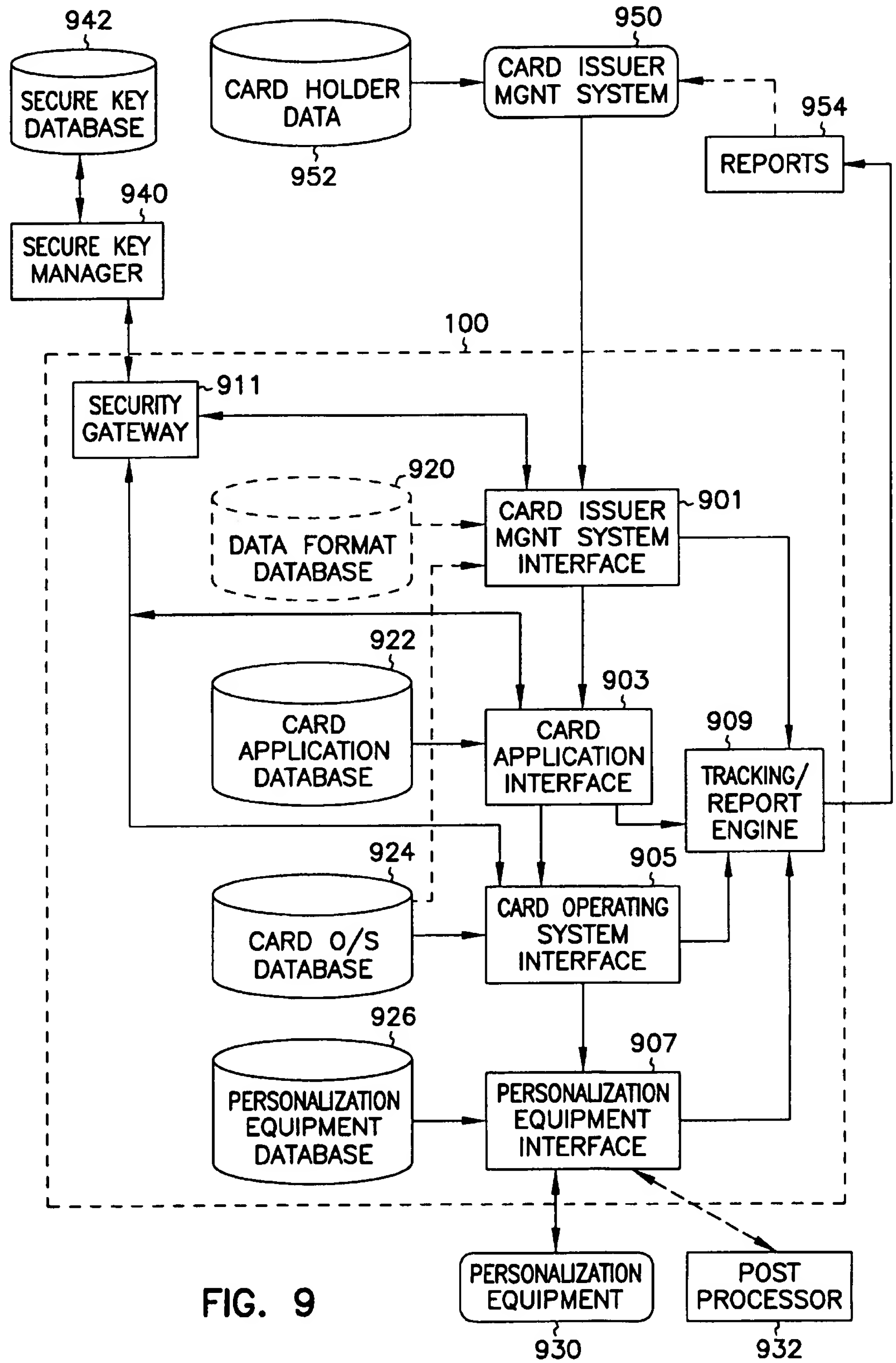
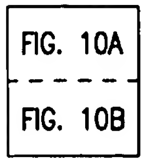

942
SECURE KEY DATABASE
CARD HOLDER DATA
952
950
CARD ISSUER MGNT SYSTEM
954
REPORTS
940
SECURE KEY MANAGER
100
SECURITY GATEWAY
911
920
DATA FORMAT DATABASE
CARD ISSUER MGNT SYSTEM INTERFACE
901
922
CARD APPLICATION DATABASE
903
CARD APPLICATION INTERFACE
909
TRACKING/ REPORT ENGINE
924
CARD O/S DATABASE
905
CARD OPERATING SYSTEM INTERFACE
926
PERSONALIZATION EQUIPMENT DATABASE
907
PERSONALIZATION EQUIPMENT INTERFACE
PERSONALIZATION EQUIPMENT
930
POST PROCESSOR
932

FIG. 9

FIG. 10



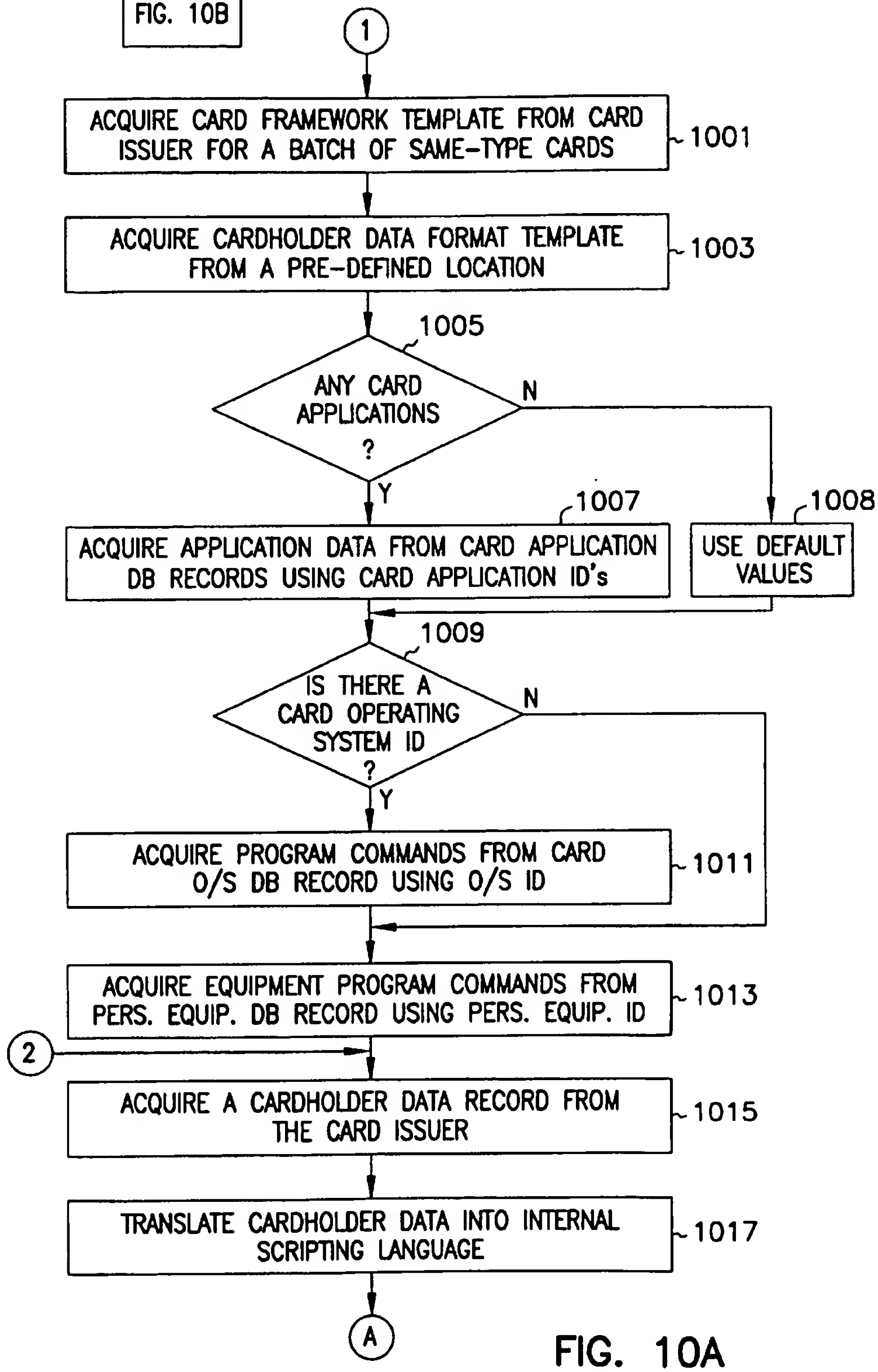


FIG. 10A

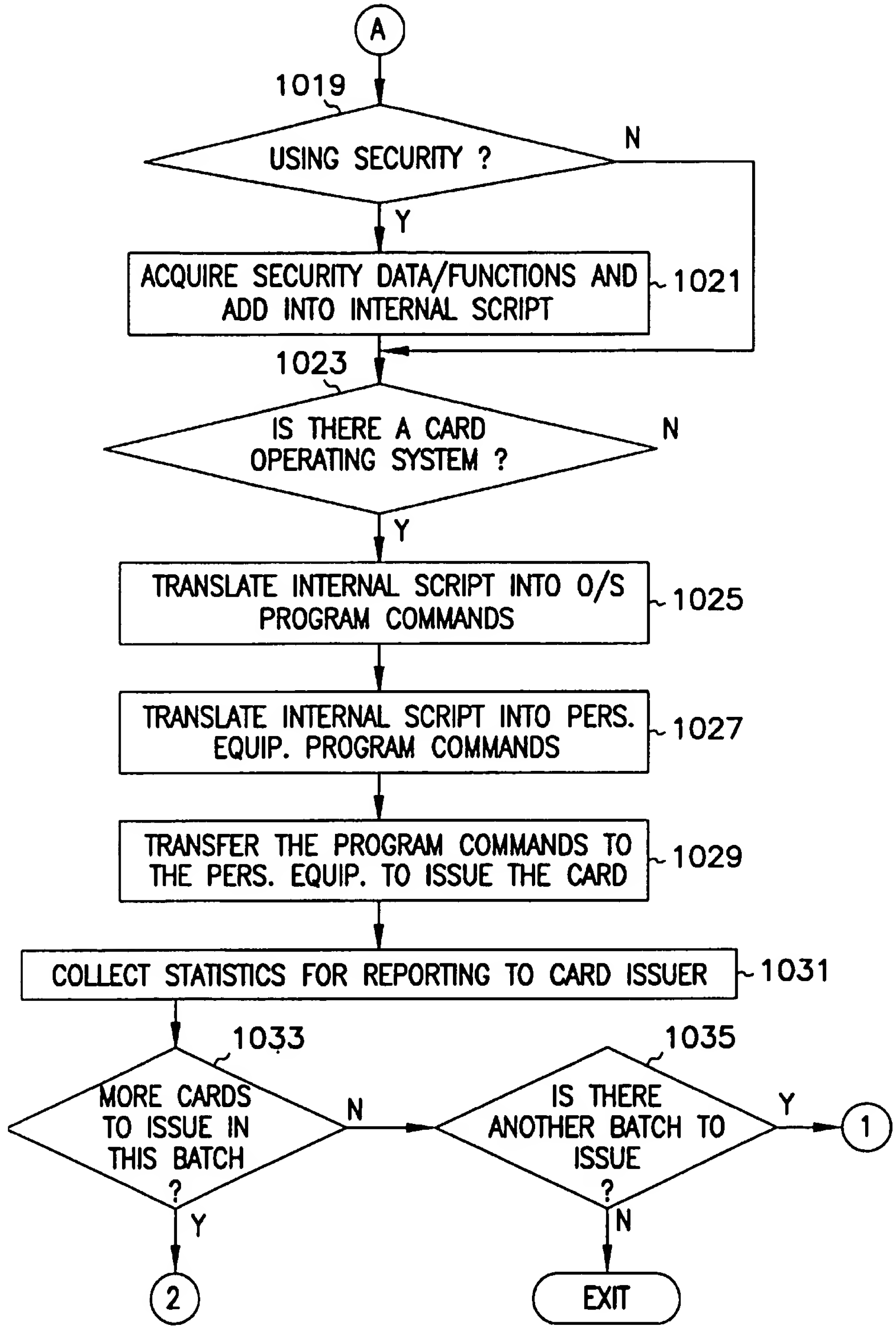

A
1019
USING SECURITY ?
N
Y
ACQUIRE SECURITY DATA/FUNCTIONS AND ADD INTO INTERNAL SCRIPT
1021
1023
IS THERE A CARD OPERATING SYSTEM ?
N
Y
TRANSLATE INTERNAL SCRIPT INTO O/S PROGRAM COMMANDS
1025
TRANSLATE INTERNAL SCRIPT INTO PERS. EQUIP. PROGRAM COMMANDS
1027
TRANSFER THE PROGRAM COMMANDS TO THE PERS. EQUIP. TO ISSUE THE CARD
1029
COLLECT STATISTICS FOR REPORTING TO CARD ISSUER
1031
1033
MORE CARDS TO ISSUE IN THIS BATCH ?
N
Y
2
1035
IS THERE ANOTHER BATCH TO ISSUE ?
Y
1
N
EXIT

FIG. 10B

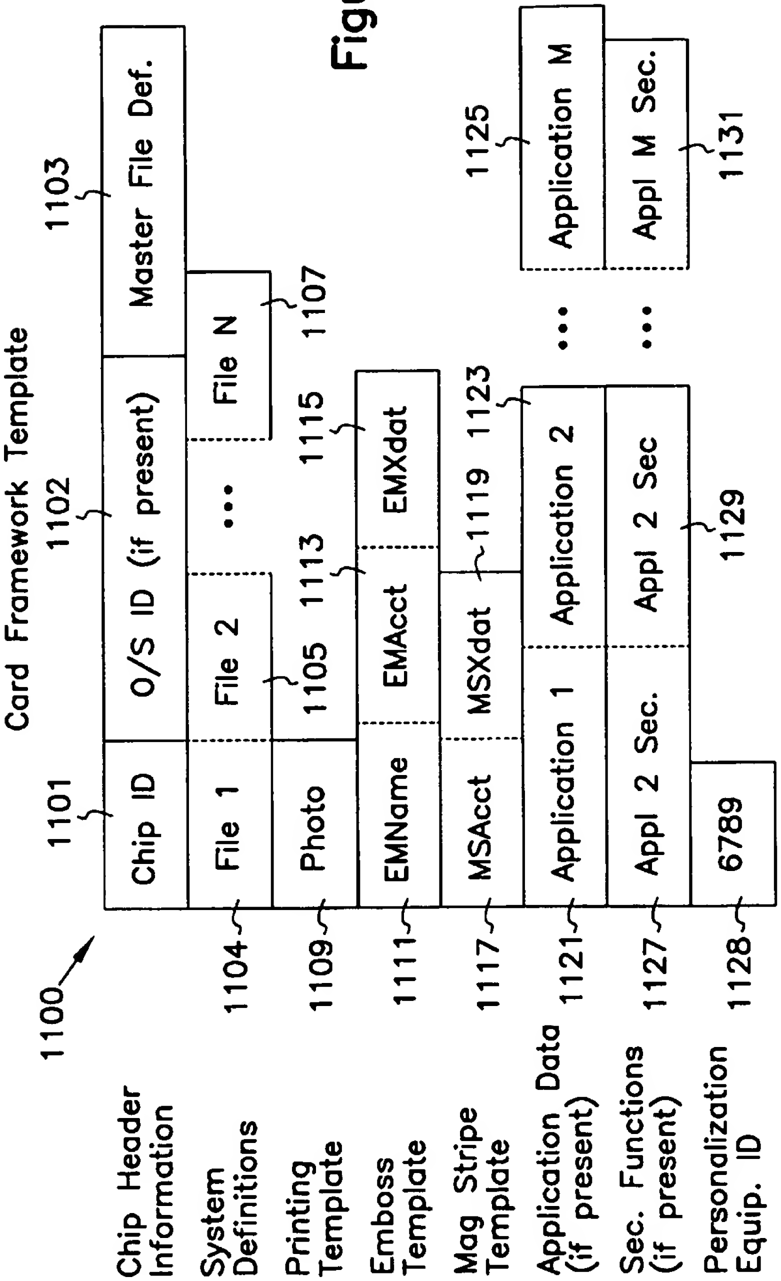


Figure 11

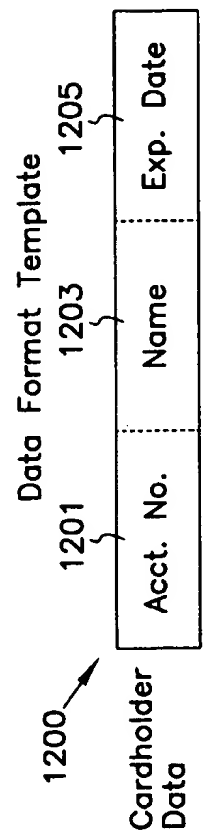


Figure 12

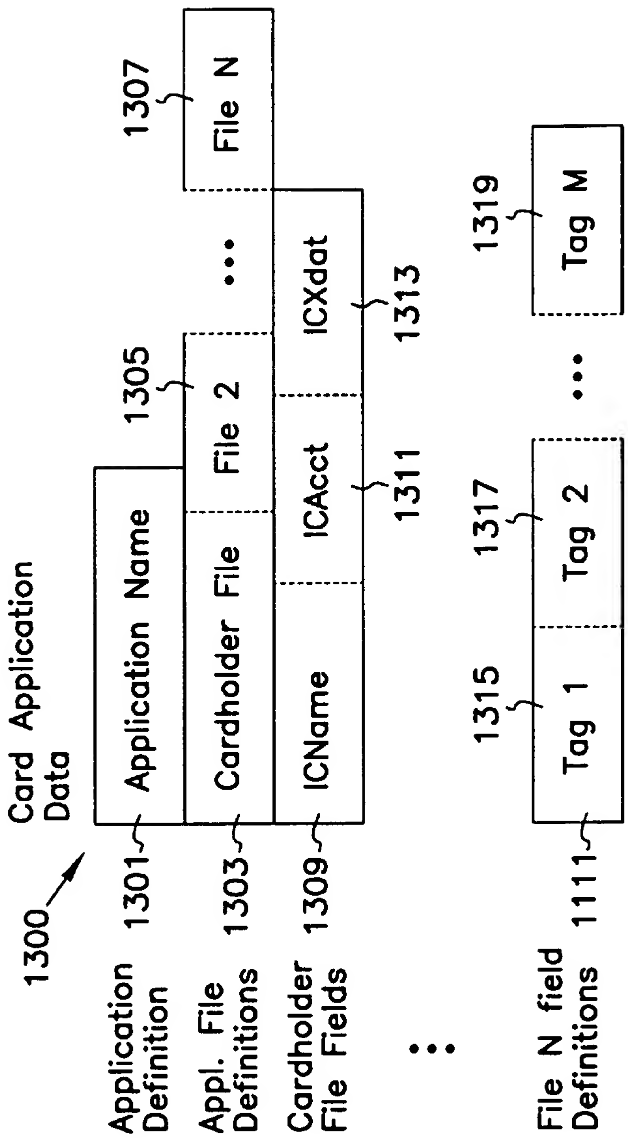

1300
Card Application Data
Application Definition
1301
Application Name
Appl. File Definitions
1303
Cardholder File
File 2
1305
...
File N
1307
Cardholder File Fields
1309
ICName
ICAcct
1311
ICXdat
1313
...
File N field Definitions
1111
Tag 1
1315
Tag 2
1317
...
Tag M
1319

Figure 13

MATERIALS STATUS

FIG. 14A

SOFTWARE CONFIGURATION

PRODUCTION STATUS

FIG. 14B

FIG. 14

SAMPLE REPORT ITEMS

MATERIALS STATUS

- CARD STOCK
  - PART NUMBER
  - ACCOUNT NAME
  - PART DESCRIPTION
    - PLASTIC
    - MICROCONTROLLER
    - MAGNETIC STRIPE
  - QUANTITY ON HAND
  - REORDER LEVEL
  - PROJECTED NEXT REORDER DATE / QUANTITY
  - INVENTORY RESTOCKING DATA
    - RECEIVED LOT NUMBER
    - VENDOR
    - DATE RECEIVED
    - QUANTITY
    - QUALITY / REJECTS
  - INVENTORY USE DATA
    - JOB NUMBER
    - DATE
    - QUANTITY CHECKED OUT
    - QUANTITY CHECKED BACK IN

- FORM STOCK
  - PART NUMBER
    - ACCOUNT NAME
    - PART DESCRIPTION
    - QUANTITY ON HAND
    - REORDER LEVEL
    - PROJECTED NEXT REORDER DATE / QUANTITY
    - INVENTORY RESTOCKING DATA
      - RECEIVED LOT NUMBER
      - VENDOR
      - DATE RECEIVED
      - QUANTITY
      - QUALITY / REJECTS
    - INVENTORY USE DATA
      - JOB NUMBER
      - DATE
      - QUANTITY CHECKED OUT
      - QUANTITY CHECKED BACK IN

FIG. 14A

SOFTWARE CONFIGURATION

- CUSTOMER LIBRARY
  - INTERFACE LIBRARY COMPONENT
- APPLICATION LIBRARY
  - KEY CARDS
  - APPLICATION DATA LIBRARY COMPONENT
- CARD O/S LIBRARY
  - BATCH CARD
  - CARD O/S LIBRARY COMPONENT
- PERSONALIZATION EQUIPMENT LIBRARIES
  - PERSONALIZATION DEVICES SUPPORTED

PRODUCTION STATUS

- MACHINE BATCH
  - BATCH ORIGINATION DATA
    - ADMINISTRATION
      - PARENT JOB
      - OPERATOR
      - DATE / TIME
      - CUSTOMER
    - CARD TYPE
      - APPLICATIONS
        - DATA
        - SECURITY REQUIREMENTS
          - STATIC
          - DYNAMIC
      - OPERATING SYSTEM
        - DYNAMIC SECURITY REQUIREMENTS
      - PLASTIC CARD TYPE (PART NUMBER)
      - NUMBER OF CARDS
    - PERSONALIZATION DEVICE
      - CHIP PERSONALIZATION DEVICE
      - CARD PERSONALIZATION DEVICE
  - PRODUCTION
      - CARDS PERSONALIZED
      - CARD ERRORS
        - ERROR TYPE
      - CARD TOTAL

FIG. 14B

SYSTEM AND APPARATUS FOR SMART CARD PERSONALIZATION

RELATED APPLICATION

This application is a non-provisional application claiming benefit under 35 U.S.C. § 119(e) of U.S. Provisional Application Ser. No. 60/015,743, filed Apr. 15, 1996.

FIELD OF THE INVENTION

The present invention is related to data storage devices and in particular to producing portable programmed data carriers such as credit cards, debit cards, identification cards, and other transaction cards.

BACKGROUND OF THE INVENTION

Increasing numbers of organizations which issue transaction cards to their users, customers, or employees require cards tailored to meet the requirements of their particular service or application. These organizations also want the cards to contain data about the card holder. Existing transaction cards encode such data in a magnetic stripe on the back of the card but the amount of data that can be held by a magnetic stripe is limited. A new type of transaction card embeds a microprocessor computer chip in the plastic of the card to greatly increase the card's data storage capacity. Additionally, sophisticated card applications specific to the card issuer can execute in certain varieties of the chips, and the chip may also contain a type of operating system. Transaction cards with embedded chips are referred to in the industry as portable programmed data carriers, more commonly called "smart cards." The chip in a smart card is programmed with initialization and/or personalization data at the same time as the surface of the card is being embossed and/or printed.

The initialization data comprises three major types of information: application data, security data, and printed data. The application data is common to all cards for a given card application and includes application program code and variables. The security data prevents fraudulent use of the card and is usually provided in the form of "secure keys." Printed data, such as a logo, bar codes, and various types of numerical information, are placed on the surface of the card. Some or all of the same data can also be embossed on the surface. Optical technology also can be employed to make part or all of the surface of the card into a storage medium with data accessible by an appropriate optical reader.

Smart cards are also programmed with information specific to an individual card holder through a process called "personalization." The personalization information for a smart card is similar to the personalization information currently contained on non-smart cards, such as the card holder's name, account number, card expiration date, and a photograph. Because of its increased storage capacity, the chip in a smart card can contain additional data beyond the basic information on the standard transaction card including a graphical representation of the individual's signature, data defining the types of service the card holder is entitled to, and account limits for those services.

The smart card issuing process must control and report on each personalized card and the results of the personalization process. Extensive report and audit files thus must be maintained to support the card tracking requirements.

Currently, a smart card issuing system must be tailored to meet the requirements of a specific card application that will be programmed on a specific type of smart card under the

control of a specific card operating system and to format the data for the card to be compatible with a specific type of personalization equipment chosen to issue the card. The entire issuing system must be re-configured whenever any one of these variables (issuer application, smart card/card operating system, and/or personalization equipment) is changed, increasing the time and cost incurred by the issuer of the card in delivering personalized smart cards to its customers. Additionally, many of the current issuing systems lack a viable means to provide dynamic feedback regarding the status of any particular batch of cards in the process to the card issuer.

Furthermore, the smart card issuing systems in use today utilize a proprietary approach developed by either the card manufacturer or the personalization equipment manufacturer. To encourage sales of their respective cards or equipment, each manufacturer develops a unique personalization solution for a particular card application, and each solution is specific to a particular card issuer. These unique solutions are intended to optimize performance of the cards or equipment and thus do not permit a more inclusive, generalized personalization process that accepts any card operating system and/or work with any personalization equipment.

As the demand for smart cards increases, a smart card issuing system which permits the card issuers to use any type of personalization equipment to handle multiple types of smart cards, and their attendant operating systems, and to embed the issuers' specific card applications along with the required card holder data in any of the various types of smart cards is required.

SUMMARY OF THE INVENTION

A smart card personalization system maintains a database containing card application data, issuer format template data, card operating system data, and personalization equipment data to permit a card issuer to dynamically change card applications, card and card operating systems, and/or personalization equipment in a card issuing process without the necessity of modifying the card issuer's interface to the issuing process.

The smart card personalization system issues portable programmed data carriers, or smart cards, by first acquiring a data format identifier, a card operating system identifier, a personalization equipment identifier, an application program identifier or identifiers, and personalization data for a card holder from a card issuer management system. The identifiers permit the system to address data stored in a data structure, such as a database, and specify the particular data needed by the system for each card to be issued. Because each card issuer formats its personalization data differently and may have multiple data formats, the smart card personalization system has a database of data format templates that enable it to interface with multiple card issuer management systems. The system acquires the format template defining the personalization data used by a particular card issuer from a record in the database identified by the data format identifier. The system uses the data format template to translate the personalization data from the card issuer's format into an internal format recognized by the components of the system. The system uses the card operating system identifier and application program identifier(s) to acquire programming control commands for an operating system pre-loaded in a microprocessor chip embedded in the card, and application data, in the form of code and/or variables, for an application program type or types from the database.

The system also acquires the equipment characteristic data for the personalization equipment to be used to issue the smart card using the personalization equipment identifier. Once the system has acquired all the data necessary to issue the smart card it transfers the programming control commands, the application code and variables, and the translated personalization data to the personalization equipment as specified by the equipment characteristic data.

Alternatively, no data format identifier is passed by the card issuer because the data format template is derived from data in the application data record or because the format of the personalization corresponds on a one-to-one basis with the internal format used by the system. The card issuer may also substitute the data format template record for the data format identifier so the system does not need to reference its database of format records.

Another feature of the smart card personalization system is its card management function. The smart card personalization system collects information regarding the card issuing process and reports this information to the card issuer management system.

Smart cards may include one or more "secure keys" that are programmed into the chip to prevent fraudulent use of the card. The appropriate secure key data is obtained by the smart card personalization system from secure key records maintained by the card issuer, or another security source, and then transferred to the personalization equipment. The security source also provides security functions that are used by the smart card personalization system to ensure the integrity and secrecy of data during the transmission of data to and from the system and within the system during the smart card personalization process.

The smart card management system performs the functions described above through a series of software modules executing on a computer or multiple computers. One module is a card issuer management system interface which acquires the data format identifier, the card operating system identifier, the personalization equipment identifier, the application program identifier(s), and the personalization data for a card holder from the card issuer management system. The card issuer management system interface then uses the data format identifier to acquire the format template that defines the personalization data and translates the personalization data into the common, internal data format. A card operating system interface module acquires the programming control commands for the card operating system type specified by the card operating system identifier. A card application interface module uses the application program identifier(s) to determine which type(s) of application program is to be placed on the card and acquires the specified application code and variables. A personalization equipment interface module is responsible for the acquisition of the equipment characteristic data for the personalization equipment type specified by the personalization equipment identifier, and further for transferring the programming control commands, the application code and variables, and the translated personalization data to the personalization equipment in accordance with the requirements stipulated by the equipment characteristic data.

The reporting and security functions are provided by a tracking/report module and by a secure key management module.

The smart card personalization system uses an underlying data structure, such as a database, residing in a computer storage medium to organize the data necessary to issue the smart cards. The data structure comprises several different types of data elements and uses "indices" or "identifiers" to quickly access specific data. There are four main data elements in the system: a data format element, a card operating system element, an application program element, and a personalization equipment element.

The data format element contains a template that defines the format of the personalization data used the card issuer. The data format element may be stored in a database containing data format elements for various card issuer and the information stored in the data format element is accessed through the data format identifier. Alternatively, the data format element may be derived at the time the card is issued from data in the application program element(s) so that the application program identifier(s) passed by the card issuer identify the data format. When the data format of the personalization data corresponds exactly to the internal format used by the smart card personalization system, the data format template is logically implied which creates a virtual data format element for the issuing process.

The card operating system element holds the programming control commands that direct the card operating systems controlling a smart card chip and is accessed through the card operating system identifier.

The application program element(s) contains application data, such as program code and variables, required by the applications associated with various card issuers; application data is accessed through an application program identifier(s).

Operating parameters for various types of personalization equipment used to issue smart cards are stored in the personalization equipment element and accessed through a personalization equipment identifier corresponding to the type of the personalization equipment to be used during an issuing run.

Special configurations of the smart card personalization system support card issuers that do not need the full flexibility of the system described above.

The smart card personalization system addresses the weakness in the prior art by providing a centralized interface of inputs and outputs to the smart card personalization process which is designed to dynamically accommodate changes in the issuing process. The system interfaces to any issuer management system, manages the transfer of card holder data and card applications to the particular personalization equipment used, and collects statistics for real-time and off-line inquiries to support critical management and reporting functions. The system maintains a database of issuer data formats, card operating systems, card application programs, and types of personalization equipment. This database enables the system to handle any combination or permutations of the data, thus improving cost and time to market for the issuer. Furthermore, the system interfaces with various card security methodologies to reduce fraud.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1A is a block diagram representing a smart card issuing process that incorporates a smart card personalization system.

FIG. 1B is a functional block diagram of input and output connections for the smart card personalization system shown in FIG. 1A.

FIG. 1C is a functional block diagram showing software modules and data structures which comprise one embodiment of the smart card personalization system shown in FIG. 1B.

FIG. 2 is the functional block diagram of the embodiment of FIG. 1C with the addition of a security module to manage keys used for smart cards.

FIG. 3 is a functional block diagram of another embodiment of the smart card personalization system showing a minimal configuration to manage multiple types of cards and personalization equipment.

FIG. 4 is the functional block diagram of the embodiment of FIG. 3 with the addition of a module to manage multiple card operating systems.

FIG. 5 is the functional block diagram of the embodiment of FIG. 4 with the addition of the security module.

FIG. 6 is the functional block diagram of the embodiment of FIG. 3 with the addition of a module to manage multiple card applications.

FIG. 7 is the functional block diagram of the embodiment of FIG. 6 with the addition of the security module.

FIGS. 8A, B represent a high level flow chart for computer software which implements the functions of the smart card personalization system.

FIG. 9 is a functional block diagram of an alternate embodiment of the smart card personalization system using software modules and data records.

FIGS. 10A, B represent a high level flow chart for computer software which implements the functions of the embodiment of the smart card personalization system shown in FIG. 9.

FIG. 11 is a data field chart for a card framework template record used by the embodiment of the smart card personalization system shown in FIG. 9.

FIG. 12 is a data field chart for a data format template record used by the embodiment of the smart card personalization system shown in FIG. 9.

FIG. 13 is a data field chart for a card application data record used by the embodiment of the smart card personalization system shown in FIG. 9.

FIGS. 14, 14A, 14B represent a report format showing sample items tracked by the smart card personalization system.

DESCRIPTION OF THE EMBODIMENTS

In the following detailed description of the embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. These embodiments are described in sufficient detail to enable those skilled in the art to practice the invention, and it is to be understood that other embodiments may be utilized and that structural, logical and electrical changes may be made without departing from the spirit and scope of the present inventions. The following detailed description is, therefore, not to be taken in a limiting sense, and the scope of the present inventions is defined only by the appended claims.

The leading digit(s) of the reference numbers in the Figures usually correspond to the figure number, with the exception that identical components which appear in multiple figures are identified by the same reference numbers.

Issuing Smart Cards

Standard transaction cards such as regular credit cards are familiar to most people. A transaction card usually has information about the card holder, such as name and account number, printed and/or embossed on the surface of the card. Transaction cards frequently contain a magnetic stripe which is encoded with card holder data as well. The process of printing/embossing/encoding the card holder data on each transaction card is known as "personalization." Each transaction card also undergoes a process known as "initialization" in which certain types of information common to all cards in a batch, such as an issuer identifier and batch number, are placed on the card.

A smart card differs from a standard transaction card in that a computer microprocessor chip is embedded in the plastic of the card to greatly increase the card's data storage capacity. In some varieties of smart cards, the card manufacturer pre-loads the chip with one of several possible card operating systems and the operating system controls the programming of the chip during the personalization process. Additionally, sophisticated card applications specific to the card issuer may execute in certain varieties of the chips.

The initialization data for a smart card comprises three major types of information: application data, security data, and printed data. The application data is common to all cards for a given card application and includes application program code and variables that are programmed into the chip. The security data, usually provided as secure keys or security functions, validates the data on the card and prevents fraudulent use of the card. Printed data, such as a logo, bar codes, and various types of numerical information, are printed on the surface of the card. Some or all of the same data may also be embossed on the surface. Optical technology also may be employed to make part of the surface of the smart card into a storage medium with data accessible by an appropriate optical reader.

The personalization information for a smart card is similar to the personalization information currently contained on non-smart cards, such as the card holder's name, account number, card expiration date, and a photograph. Because of its increased storage capacity, the chip in a smart card may contain additional data beyond the basic information on the standard transaction card including a graphical representation of the individual's signature, data defining the types of service the card holder is entitled to, and account limits for those services.

Smart Card Personalization System

FIG. 1A shows components of a smart card issuing process that incorporates an embodiment of the smart card personalization system of the present invention. The smart card personalization system **100** receives data from a card issuer management system **150** (typically proprietary to the card issuer), translates the data into a data stream, and outputs the data stream to personalization equipment **130** which personalizes the smart cards **160**. The card issuer management system **150** manages the card holder data and determines the type of card to issue, the card applications to embed in the card, and what personalization equipment to use to issue the card for a particular card holder. The card issuer management system is frequently a computer program as illustrated in FIG. 1A, but the smart card personalization system **100** is capable of receiving data from alternate inputs, such as a person inputting the data from a telephone keypad.

The smart card personalization system **100** is illustrated in FIG. 1A as a software program executing in a computer. As described below, the smart card personalization system **100** accesses database records which define various types of cards and card operating systems, card applications, and personalization equipment. The logical functions of the software and the database may be distributed among computers in a client/server network or centralized into a single processor. The functions may also be distributed across

processors connected through standard local area networks, wide area networks, dedicated phone lines or other communication means used to loosely couple processors. The software program executes under an operating system such as Unix, Windows 95©, or Windows NT©, and on industry-standard workstation and/or personal computer hardware.

The system 100 controls card printers, embossing devices, and integrated or add-on smart card interface devices collectively represented in FIG. 1A as personalization system 130. Personalization equipment 130 also represents such devices as large volume card printer/embossers, small volume card printer/embossers, automatic teller machiners (ATMs), point of sale terminals, unattended kiosks, personal computers, network computers, and on-line telecommunication devices. Because of their investment in existing non-smart card personalization equipment, many card issuers do not purchase entirely new smart card personalization equipment but instead augment their existing personalization equipment with a smart card interface device which programs the chip in the card while the older device performs the printing and embossing functions. In such a configuration, the computer system executing the smart card personalization system 100, or "host," may be physically connected to both devices or to only one of the devices. In the latter case, the host controls the directly-connected device and has a logical connection to the other. The physical connection between the devices and the host varies according to the manufacturer and model of the device. Common industry standard connections include serial RS232, SCSI (Small Computer System Interface), Ethernet, and serial TTL (Transistor-Transistor Logic). In addition, some devices require a proprietary bus connection.

The connections between the smart card personalization system 100 and the card management system 150 and the devices 130 may also be implemented through standard local area networks, wide area networks, dedicated phone lines, or other remote communication infrastructure used to transfer data. The use of such remote connections when personalizing smart cards is described in U.S. Pat. No. 5,524,857 issued on Jul. 9, 1996, to Laing, et al. Alternate connections will be apparent to those skilled in the art and are within the scope of the invention.

FIG. 1B is a block diagram of one embodiment of the smart card personalization system illustrating the logical connections between the smart card personalization system 100 and functions employed by a card issuing organization to issue smart cards. Card holder data maintained by the card issuing organization contains information about each individual card holder, such as name, account number, card expiration date, and applicable services. Various ways of inputting the card holder data into the card issuer management system 150 are shown in phantom as card holder data 152 in FIG. 1B. The card issuer management system 150 may receive the card holder data on computer media, such as magnetic tape, floppy disk, or CD ROM. Alternatively, the card holder data 152 may be input through an on-line connection such as a general switched telephone network, a packet-switched network, i.e., the Internet, a dedicated line, or a cable/satellite television signal. Additional ways in which the card holder data 152 may be input to the system 150 will be apparent to those skilled in the art.

In addition to the card issuer management system 150, the card issuer typically has an existing reporting capability 154 with which the smart card personalization system 100 interfaces so that the card issuer can review statistical information maintained by the system 100. An external security source, also provided by the card issuer and shown as secure key manager 111 and secure key database 128, provides security functions that work in conjunction with the card issuer management system 150 and the smart card personalization system 100. FIG. 1B also illustrates an alternate embodiment of the smart card personalization system 100 which supports a card issuer that has add-on smart card interface devices. The system 100 directs a portion of the personalization information to the older personalization equipment 130 and the remainder of the data to a post-processor 132 in the smart card interface device 132 which programs the chip. These functions are explained in detail below.

The embodiments of the software program for the smart card personalization system 100 shown in the following Figures function as combinations of code modules with each module executing a specific part of the issuing process. In these embodiments, the modules are coupled through defined input and output program calls, and are also coupled to the data structures through standard data query commands that provide access to the data stored in the data structures. The communication protocols between the modules, and between the modules and the data structures vary depending on the language in which the modules are written and upon the underlying data management system employed to support the database.

FIG. 1C is a more detailed functional block diagram of the smart card personalization system 100 of FIG. 1 B without the external security functions. FIG. 1C shows the internal connections between software modules and database records that enable the smart card personalization system 100 to combine multiple types of issuer data formats, card operating systems, card applications and personalization equipment when issuing smart cards.

The smart card personalization system 100 provides a customized card issuer management interface 101 to a card issuer management system 150. In this embodiment, the card issuer management system 150 passes personalization data from a card holder database 152 to the system 100. Each software module within system 100 expects the personalization data to be passed to it in a particular, internal format. Because the personalization data is in an external format defined by the card issuer that often differs from the internal format(s) expected by the software modules, the personalization data is translated by the system 100 into the internal format(s) using the data format template. The system 100 may acquire the data format template through a data format identifier passed by the card issuer that the system 100 uses to acquire an optional data format template record 120 (shown in phantom in FIG. 1C) as illustrated by an optional connection between the record 120 and the card issuer management system interface 101. Alternatively the card issuer passes the data format template record to the system 100 instead of the data format identifier. In another embodiment, the data format template may be derived from the data in the card application record 124 that is specified by an application program identifier passed by the issuer as illustrated by an optional connection between the card application database 124 and the card issuer management system interface 101.

In a further alternate embodiment of FIG. 1C, security functions are provided internal to the smart card personalization system 100, by passing security functions into the system as part of the card application record.

A further alternate embodiment in which the personalization data format matches the internal format is also shown in FIG. 1C. Because no translation between the external and internal formats is necessary in this embodiment, no data

format template is needed so the data format record **120** and the connections between the card issuer management system interface **101** and the data format record **120** and the card application database **124** are not present. The data format record may **120** be composed of a plurality of tables which instruct the system **100** as to the proper parsing of the personalization data or a simple list that indicates the order in which the fields of the card holder data record appear as will be apparent to those skilled in the art. The various alternate procedures for determining the format of the personalization data described above are implicit in all the embodiments of the smart card personalization system **100** described herein.

Using a card identifier provided by the card issuer management system **150**, a card operating system interface module **103** retrieves programming control commands specific to the card operating system **122** for the microprocessor chip that is embedded in the type of card being issued. The programming control commands direct the encoding of the chip with the personalization data and the card application(s) chosen by the card issuer.

Each card application comprises program code and variable data that is stored in the database as application data **124** and is identified by an application program identifier. The card issuer management system **150** passes one or more program application identifiers to the system **100** which are used by a card application interface module **105** to acquire the corresponding application data **124**.

The personalization equipment that the card issuer plans to use to issue the batch of cards is defined by a personalization equipment identifier. A personalization equipment interface module **107** acquires equipment characteristic data **126** specific to the type of personalization equipment **130** corresponding to the personalization equipment identifier. The personalization equipment interface **107** also acquires the programming control commands, the application code and variables, and the translated personalization data, and transfers all of this data to the personalization equipment **130** as specified by the equipment characteristic data **126** to issue the smart card.

An alternate embodiment of the system **100** supports a card issuer that has augmented their existing personalization equipment with a smart card programming device by having the personalization equipment interface **107** direct a subset of the translated personalization information to the older personalization equipment **130** and the remainder of the data to a post-processor **132** in the smart card programming device.

The smart card personalization system **100** also provides a tracking/report module, or engine, **109** that collects statistical information from the other modules in the system **100** and formats the statistical information for output as hard-copy reports **154** or as input to a reporting function in the card issuer management system **150**. Because this statistical information is being gathered in real-time, the card issuer management system **150** can interactively query tracking/report module **109** to obtain statistics about the smart card personalization system as it is executing. Examples of items monitored by the tracking/report module **109** are shown in FIG. **14**.

In an alternate embodiment shown in FIG. **2**, the smart card personalization system **100** includes a security source in the form of a secure key manager module 111 and secure key database **128**. When a smart card is manufactured, the vendor includes security architecture on the chip to prevent unauthorized programming. The security architecture implementation is commonly dependent on the application(s) programmed onto the chip. For example, the secure keys programmed in a stored value application would be different than the secure keys programmed in a health care application. The security architecture implementation also varies depending on the type of card: some cards require a single secure key which enables chip programming while others require multiple secure keys to enable chip programming and to perform additional security functions. FIG. **2** illustrates the basic functions of the secure key manager **111** when interfacing with the security architecture on a card that requires multiple secure keys.

As shown in FIG. **2**, the secure key data is stored in the secure key database **128** which is external to the smart card personalization system **100** and maintained by the card issuer or other security source. Extending the secure key manager **111** to handle more or fewer secure keys, and to interface with a secure key database managed by the smart card personalization system **100** itself, is dependant on the application, operating system, and personalization equipment being used in the specific card issuing application, and will be apparent to those skilled in the art.

The secure key manager **111** also provides additional mechanisms to ensure secure key data authentication, data integrity and data secrecy. In one embodiment, secure key data authentication is accomplished through the implementation of various encryption methods. Secure key data integrity is achieved through digital signature mechanisms that use public keys to ensure that secure key data is being transmitted and received from valid sources. Secure key data secrecy is ensured by encrypting the transmitted data with a private key that is shared with the data receiver and which the data receiver uses to decrypt the data upon receipt.

After the system **100** receives a secure key record from the secure key database **128**, the secure key manager **111**, in conjunction with the card operating system interface **103** and the card application interface **105**, perform the secure key authentication, data integrity and data secrecy functions. The system **100** then transfers the secure key data to the personalization equipment **130** through the personalization equipment interface **107** along with the other data for the card.

In an alternate embodiment, the secure key manager **111** passes security information to the other modules of the smart card personalization system **100**. For example, portions of the card holder data, such as the PIN (Personal Identification Number) code, may be encrypted by the card issuer management system **150** prior to passing the data to the smart card personalization system **100**. The card issuer management system interface **101** retrieves the encryption key from the secure key database **128** through the secure key manager **111**, and decrypts the data prior to encoding or programming the PIN code into the magnetic stripe and/or the chip.

In a further alternate embodiment, the secure key manager **111** is a code "hook" into the smart card personalization system **100** which provides a gateway connection for an external security source that supplies the required security functions. An example of such an external security source is a security manager program written by a third party that manages a security database of secure keys and/or security functions similar to secure key database **128**. The security functions may be either external routines executed by the security manager, or code modules passed by the security manager which are then executed by the smart card personalization system **100** to provide the required security functions, or a combination of both.

FIG. **3** illustrates a minimal configuration of the smart card personalization system **100**. In this embodiment, only

the card issuer management system interface modules **101** and the personalization equipment interface modules **107** are enabled in the software. This embodiment permits card issuer to use the system **100** to personalize non-smart cards, thus saving the cost of having two separate personalization systems, while permitting the card issuer to use multiple data formats and multiple types of personalization equipment. FIG. **3** also illustrates an additional alternate embodiment that includes the tracking/report module **109** as described above in conjunction with FIG. 1C.

In a still further alternate embodiment, the smart card personalization system **100** shown in FIG. **3** encodes data onto an optical transaction card when optical-encoding equipment is used as the personalization equipment **130**.

FIGS. **4** and **5** depict still further alternate embodiments that are implemented when the card issuer does not program a card application on the smart card chip. These embodiments allow the card issuer to issue multiple card types with their attendant variety of operating systems on multiple types of personalization equipment without having to reconfigure the smart card personalization system **100**. As described above in conjunction with FIG. 1C, FIG. **4** includes the modules that support reporting and post-processing. FIG. **5** illustrates the embodiments of FIG. **4** with the addition of the secure key manager module **111** that provides security to the card operating system interface **103** for transmission to the personalization equipment **130**.

Similarly, FIGS. **6** and **7** illustrate embodiments to support a card issuer that uses the chip on a smart card only as a data storage device for a card application, and so does not have an operating system executing on the chip. Smart card personalization system **100** supports multiple card applications for multiple card types issued with multiple types of personalization equipment. FIGS. **6** and **7** are analogous to FIGS. **4** and **5** except that the secure key manager **111** provides secure keys and/or functions to the card application interface **105** instead of the card operating system interface **103**.

FIG. **8** is a high level flow chart for one embodiment of software which implements the functions of the smart card personalization system **100** described above. The software acquires a personalization equipment identifier for a batch of transaction cards to be issued from the card issuer management system at block **801**. Depending on the type of cards to be issued, the software also acquires a program application identifier(s) and/or a card operating system identifier at the same time. The software then acquires the particular data format template corresponding to the format of the personalization data through one of the procedures described above (block **803**). At block **805**, the system acquires the equipment characteristics for the personalization equipment to be used to issue the batch of cards from the personalization equipment record specified by the personalization equipment identifier.

If a card operating system identifier was passed by the card issuer management system (block **807**), the software retrieves the programming control commands from the card operating system database record corresponding to the card operating system identifier at block **809**. Blocks **811** and **813** perform the same logic for a card application, retrieving the application data, such as code and/or variables, from the database. At this point, the software has acquired the common data necessary for all the cards in the batch and begins looping through the logic which issues cards for the individual cardholders.

The card issuer management system passes the personalization data for a single card holder to the software (block **815**) which translates the data items from the format defined by the data format template into an internal format used by the modules of the smart card personalization system (block **817**). If the card chip contains security architecture that requires secure keys (block **819**), the software acquires the secure key data necessary to perform the secure key functions from the appropriate secure key source at block **821**.

The software is now ready to transfer data to the personalization equipment to program the card. If the card is protected by secure keys, the secure key functions are performed and the secure key data is transferred at block **823**. Then the programming control codes for the chip operating system, if applicable, are transferred (blocks **825** and **827**); next the application code and/or variables are transferred if they are needed (blocks **829** and **831**). Finally, the card holder's personalization data that was translated into the internal format is transferred (block **833**).

After the data has been transferred to the card, the software adds the appropriate values to the statistics it collects for the card issuer management system at block **839**. If more cards in the same batch remain to be issued (block **841**), the software returns to block **815** and acquires the personalization data for the next card holder. Otherwise, the software determines if the card issuer management system has a different batch of cards to issue (block **843**) and returns to block **801** to acquire the necessary information to repeat the cycle for the new batch. If no further cards are to be issued, the software exits.

The mechanisms by which the card issuer management system **150** passes the necessary data to the smart card personalization system **100** and the order in which the smart card personalization system processes the data from the card issuer management system may be changed without exceeding the scope of the invention. Different arrangements are dictated by the specific environment in which the system **100** operates as shown in the alternate embodiment illustrated in FIGS. **9** and **10**.

In FIG. **9**, a security module **911** acts as a gateway into the smart card personalization system **100** for a security source such as security manager **940** and security database **942** shown in FIG. 1B as **111** and **128** respectively. The security manager **940** controls access to the security database **942** and connects into the security gateway **911** to perform the necessary security functions for the smart card personalization system **100**. The security gateway **911** is coupled to the card issuer management system interface **901** which allows the interface **901** to request that the security manager **940** decrypt personalization data passed in an encryption format by the card issuer management system **950**. The security gateway **911** is also coupled to the card application interface **903** and the card operating interface **905** so that it can supply the necessary secure keys and/or security functions to those interfaces as explained above in conjunction with FIG. **2**.

Furthermore, the embodiment of the smart card personalization system **100** shown in FIG. **9** acquires the application data **922** specified by the application program identifier prior to acquiring the programming control commands specific to the card operating system **924** using the card identifier. This embodiment permits the personalization data and the application data to be translated into the internal format prior to retrieving the programming commands for the card operating system **924** and the equipment characteristic data **926**, thus speeding the processing of each smart card.

Standard transaction cards have data printed and embossed on the surface of the card and/or data encoded in a magnetic stripe on the card. With a smart card, data may also be stored in an internal memory area within the micro-

processor. The same data may be placed on the surface of the card, in the magnetic stripe and also in the chip memory. The exact configuration of the data in and on the card will vary depending on the type of smart card being issued and the requirements of the card issuer.

FIG. **10** is a high level flow chart of the embodiment shown in FIG. **9** and, in conjunction with FIGS. **11**, **12** and **13**, further illustrates how different mechanisms may be used to implement the smart card personalization system **100**. The card issuer management system **950** passes a card framework template that defines the configuration of the smart card to the smart card personalization system **100** at block **1001**.

FIG. **11** illustrates one embodiment of the data layout for the card framework template record **1100**. The microprocessor chip identifier **101** and the card operating system identifier **1102** (if present) are specific to the type of smart card to be issued. The master file definition **1103** contains control information such as the chip source and the last date the chip was altered. The system file definitions **1104**, **1105**, **1107** contain addresses for the location of the system files within the memory of the chip. The system files are used by the card operating system and contain information such as the PIN code(s) for the card and applications, and algorithm tables. In the embodiment shown in FIG. **11**, the master file and the system file definitions conform to the International Standards Organization (ISO) directive number 7816-4.

The next three sections of the card framework template record **1100** define the arrangement of data on the surface and magnetic stripe of the card. If information is to be printed on the card, such as the card holder's photograph **1109**, the location on the surface of the card to print such data is passed by the card issuer management system **950** in the printing template of the card framework template record **1100**. Similarly, the locations on the surface of the card to emboss data is passed in the emboss template, and the arrangement of the data to be encoded in the magnetic stripe is passed in the mag stripe template. The emboss data is illustrated in the card framework template record **1100** as the card holder's name (EMName) **1111**, account number (EMAcct) **1113**, and expiration date (EMXdat) **1115** and the magnetic stripe data by the account number (MSAcct) **1117** and the expiration date (MSXdat) **1119**. The number of data items in the printing, emboss, and mag stripe templates will vary depending on the configuration of the smart card desired by the card issuer as will be apparent to those skilled in the art.

If the card issuer wants card applications programmed into the chip in the smart card, the card issuer passes the application program identifiers to the smart card personalization system **100** in the sections **1121**, **1123**, **1125** of the card framework template record **1100**. Each application may have specific security functions associated with it (**1127**, **1129**, **1131**) and that information is also passed by the card issuer management system **950**. The card framework template record **1100** also contains the personalization equipment identifier **1123** for the personalization equipment to be used to issue the smart cards.

In an alternate embodiment, the smart card personalization system **100** stores commonly used card framework template records in an internal database so that the card issuer management system **950** needs to pass only a card framework template identifier that identifies which card framework template record is to be used for a particular batch of cards.

The smart card personalization system **100** acquires the data format template for the personalization data from a pre-defined location specified by the card issuer at block **1003**. If the card issuer has passed a data format identifier to the system **100**, the data formate template record corresponding to the data format identifier is retrieved from the data format database **920**. Alternatively, the card issuer may pass the data format template record itself. When neither the data format identifier nor the data format template record is passed to the system **100**, the format of the personalization data is determined by the card application data as explained in more detail below.

An example of a data format template record is shown in FIG. **12**. The data format template record **1200** defines an hypothetical layout of the personalization data records in the card holder database **952** in which the account number **1201** is the first field, the card holder's name **1202** is the second field, and the expiration date of the card **1205** is the third field. In one embodiment, the personalization data records are commadelimited records so no data field lengths are necessary to define the record format. Thus the data format template record **1200** shown in FIG. **12** completely defines the structure of the following example of a comma-delimited personalization data record to the smart card personalization system **100**: 133444999922,Mary Jane Smith, 0299.

The smart card personalization system **100** acquires the application data for the card application, or applications, **922** corresponding to the application program identifiers, if any, that were passed by the card issuer management system **950** at block **1007**. If no application program identifiers are passed, the smart card personalization system **100** acquires default application data (block **1008**). The default and/or the application data in the card application data record(s) corresponding to the application program identifier(s) are inserted into the corresponding sections, i.e., **1121**, **1123**, **1125**, of the card framework template record **1100**.

One embodiment of the layout of a card application data record is shown in FIG. **13**. The first field in the card application data record **1300** is the application name **1301**. As with other computer-based application programs, a card application processes data from external sources such as an automatic teller machine or internal sources such as data files encoded into the microprocessor's memory. Using the smart card causes the appropriate application to be executed by the microprocessor and the application, in turn, accesses the internal files to retrieve or store data. To access internal data, the card application data record contains pointers to application files in the chip memory (**1302**, **1305**, **1037**) and also the location of fields within the application files. Some of the fields are initialized with data from the card holder database **952** when the card is issued. The application data **1300** includes an address **1303** to a card holder file located in the chip memory and defines the card holder file as containing three fields: the card holder's name (ICName) **1309**, the account number (ICAcct) **1311** and the expiration date (ICXdat) **1313**. Additional internal data is stored in other application files and the layout of those additional files is also defined by the application data **1300**.

If the chip embedded in the smart card contains an operating system as specified by the card framework template record, the smart card personalization system **100** acquires a set of programming control commands for the operating system from the card operation system database **924** at block **1011**. The programming control commands for each operating system includes commands for functions such as creating and accessing files in the memory of the chip, reading and writing records in the files located in chip **25** memory, along with security commands that authenticate PIN (Personal Identification Number) codes and control transactions that change monetary amounts stored in the chip.

The smart card personalization system **100** acquires the equipment characteristic data corresponding to the personalization equipment identifier in the card framework template record from the personalization equipment database **926** at block **1013**. Included in the equipment characteristic data is a set of personalization programming control commands which control the operation of the personalization equipment. As is the case with the card operating systems, the personalization control commands are proprietary to the vendor of the equipment but typically include commands directed to the administration, formatting, and production of smart cards.

When the smart card personalization system **100** has acquired all the data necessary to define a smart card, it is ready to accept personalization data records **952** from the card issuer management system **950**. As each personalization data record **952** is passed at block **1015**, the smart card personalization system **100** uses the data format template, if present, to translate the personalization data into an internal format, and the card application data and card framework template to map the personalization data into variables in a command script written in an internal scripting language at block **1017**. The translation and mapping process is described further below. Alternate embodiments which use a standard programming language such as Basic, Java or C instead of the internal scripting language are within the scope of the invention.

The smart card personalization system **1019** checks for security requirements for the various components of the smart card issuing process. In the embodiment of the card framework template shown in FIG. **11**, the security requirements for the applications are specified by the card framework template record **1100** at block **1019**. If there are security requirements, the smart card personalization system **100** acquires secure data and/or functions from the security manager **940** and adds the functions into the internal script at block **1021**. An alternate embodiment of the smart card personalization system **100** passes the identifiers of the card operating system and the personalization equipment, as well as the application program identifier, to the security manager **940** which retrieves the appropriate security data and/or functions from the security database **942**. The security functions typically use data from additional sources, including data stored in internal chip files, personalization data **952**, the operating system database **924**, the card application database **922**, combined with the algorithm tables stored in the chip or from an external security module, such as the security manager **940**, to perform the secure key authentication, data integrity, data secrecy and other security processes described above in conjunction with FIG. **2**.

Once the internal command script is completed, it must be translated into the proprietary programming control commands native to the card operating system (if present) and to the personalization equipment so that the personalization data is transferred to the smart card. In this embodiment, the translation is performed by a script language interpreter at blocks **1025** and **1027** using the information acquired from the card operating system database **924** and the personalization equipment database **926**.

At block **1029**, the smart card operating system **100** passes the interpreted script to the personalization equipment which then executes the programming control commands to emboss/print, encode and program the appropriate personalization data onto the surface, and into the magnetic stripe and chip respectively of the smart card. As before, if the card issuer has elected to purchase an add-on smart card programming device to attach to its existing personalization equipment, an alternate embodiment of the smart card personalization system **100** directs the control commands for the embossing and encoding to the personalization equipment **930** and the control command for the chip to the post-processor **132** in the smart card programming device.

When the issue process has been completed for one card, the smart card personalization system **100** acquires the next personalization data record if there are additional cards of the same type waiting to issue (block **1033**). Otherwise, the smart card personalization system determines if there is another batch of smart cards of a different type waiting to issue (block **1001**) and begin the issuing process again by acquiring a new card framework template record from the card issuer.

The following example uses sample data to further describe the processing performed by the embodiment of the smart card personalization system **100** shown in FIGS. **9** and **10**. The card issuer management system **950** requests the initiation of the issuing process by sending the smart card personalization system **100** a card framework template record, application program identifier(s), a card operating system identifier, a personalization equipment identifier, and optionally a data format template identifer or a data format template record. In this example, the card issuer management system **950** passes an application resource template record shown below that contains the identifiers The system **100** acquires a data format template using one of the procedures specified above and explained in more detail below in conjunction with the sample card holder data records.

Application Resource Template Record
[A1] DFT=CARD1.DFT CAT=CARD1.CAT CID=CHIPX.CID CPT=CARD1.CPT SOURCE=A1

The first statement in the record marks the beginning of information for a particular application, in this case application "A1". The next four statements define the identifiers for the card framework template record (DFT), the card application record (CAT), the card operating system record (CID) and the personalization equipment record (CPT). The final statement is the name of a file created by the card issuing management system **950** that contains the card holder data record(s). The card issuing management system **950** inputs the card holder data as either a single request or a 'batch' of requests for cards to be issued.

The system **100** retrieves the records corresponding to the identifiers from the database. The system **100** then uses the information contained in the card framework template and data format template to set up an internal "script," which it later interpretes into the specific commands contained in the card operating system and personalization equipment records that instruct the personalization equipment to process the personalization data and issue the card for each card holder.

Two sample card holder data records **952** are shown below.

Cardholder Data Records
Smith,James^12653683091245^0998^041052^mmmm
Anderson,Sue^39485003984138^0297^110248^mmmm

In these records, the format defined by the card issuer places the account name (card holder name) in the first field followed by the account number, expiration data, date of birth, and medical data.

The system **100** uses the data format template to interpret each card holder data record **952** as it is processed. The system **100** also uses the data format template and card application records **922** to validate the data **952** ensuring proper data and format. An example of a data format template corresponding to the format of the sample card holder records shown above is shown in the first line of the table below. The James Smith personalization data record is included in the table to show the correspondence between the data format template and the fields of the card holder data record. The data format template equates each field in the card holder record with an internal label, %1, %2, etc., which corresponds to the internal order used within the system **100**.

Data Format Template Record
| %1 | %2 | %3 | %4 | %5 |
Smith,James^12653683091245^0998^041052^mmmm

The example shown above represents the simplest case in which the fields of a card holder data record **952** are arranged in the internal order used by the smart card personalization system **100**. This one-to-one correspondence means that the system **100** does not have to translate the card holder data fields into the internal field order. In such a case, the data format template record is unnecessary. Thus, in a further alternate embodiment, the card issuer does not pass a data format identifier to the smart card personalization system **100**, but instead passes an indicator, such as a flag, which informs the system **100** that no data format template is needed because the card holder data fields are in a one-to-one correspondence with the internal field order. The system **100** acts on the indicator by bypassing the translation step.

A more complex example shown next is one in which the fields of the card holder data record **952** and the data within the fields are out of order relative to the internal system order. In this case, translation is necessary.

Cardholder Data in Issuer Format
1234567891245 James Smith 0998 041052 mmmm
Cardholder Data Translated into Internal Format
Smith,James^12653683091245^0998^041052^mmmm

The system **100** uses the data format template to translate the data fields into the internal order as shown above. The translation may result in the physical rearrangement of the data fields or may be a logical rearrangement in which the data format template is invoked as a key each time a field from the card holder data record is referenced by the system **100**. Various data format templates designed to translate different arrangements of card holder data will be apparent to those skilled in the art as will the substitution of tables of field equivalences or a set of parsing instructions or other mechanisms for the simple table used above to illustrate this example.

The card framework template record describes the structure of the chip on the card. In the sample shown below, the $MF entry defines a root directory (3F00), while $DF entries define a medical application (5F20), and an accounting application (5F10). Within each directory are application-specific files defined by $EF entries, such as 6F00 containing the account name and 6F10 containing the account number. All file descriptive data resides in the card framework template and is referenced at various times during the smart card issuing process.

Card Framework Template Record
$CHIP=3102,MEM=8192,SIZE=N10
$MF PATH=x3F00,TAG=ROOT,TITLE='Root Directory',SIZE=D7194
$DF PATH=x3F005F10,TAG=ACCT,TITLE='Acct Data', SIZE=D2048
$DF PATH=x3F005F20,TAG=MED,TITLE='Medical',SIZE=D1024
$EF PATH=x3F003100,TAG=ICCID,TITLE='Issuer ID',FORMAT=T,SIZE=D10
$EF PATH=x3F005F205E00,TAG=MED1,TITLE='Medical profile',FORMAT=T,SIZE=D80
$EF PATH=x3F005F106F00,TAG=NAME,TITLE='Acct Name',FORMAT=T,SIZE=A30
$EF PATH=x3F005F106F10,TAG=ACCTID,TITLE='Account No.',FORMAT=T,SIZE=N14
$EF PATH=x3F005F106F20,TAG=EXPIRE,TITLE=Expire Date',FORMAT=T,SIZE=N4
$EF PATH=x3F005F106F30,TAG=BIRTH,TITLE='Account Holder Birthdate',FORMAT=T,SIZE=N6

The card application record **922** "maps" the card holder data **952** to the data fields used by the application. The sample card application record **922** shown below has its data entries arranged in the sequence in which they are processed by the smart card personalization system **100**.

Card Application Record
$VL ICCID VALUE=1234509876
$VL MED1 %5,TYPE=A
$VL NAME %1,TYPE=A
$VL ACCTID %2,TYPE=N
$VL EXPIRE %3,TYPE=N
$VL BIRTH %4,TYPE=N
$VL FMTACCT %2(1–4)-%2(5–9)-%2(10–14)

The ICCID entry contains the chip identifier. Each of remaining entries, except for FMTACCT, maps a "tag" to the field in the card holder data record **952** that contains the information (as defined in the data format template shown above) and specifies the type of data in the field. Thus, the MED1 tag represents the fifth field in the card holder data record **952** and the data is in alpha format. The FMTACCT entry breaks the second field in the card holder data record **952**, i.e., the account number, into sections and inserts hyphens between the sections.

The card operating system record **924** contains the programming control commands necessary to program the chip on the card. The sample card operating system programming control commands shown below are taken from the ISO directive number 7816-4 and are not the internal proprietary commands of any particular card operating system.

Card Operating System Record
SELECT A0A4000002%F
WRITE A0D0%O%L%D
READ A0B0%O%L%D
RESET VALUE=xFF

Each entry in the example record above contains a tag followed by the corresponding command in the native language of the card operating system. Variable parameter fields are indicated by "%" followed by a letter and are filled in with the appropriate card holder data as each individual card is processed.

The personalization equipment record **926** contains personalization equipment characteristic data, such as instructions that define the actual sequence and steps necessary to issue a complete card on a specific set of personalization equipment. The sample instructions used in this example are fictitious and do not represent the internal proprietary instructions for any particular personalization equipment.

Personalization Equipment Record

```
$EMBOSS
    #EMB#%FMTACCT%`%NAME%
$ENCODE
    #ENC#%%%ACCTID%`%NAME%
$IC
    #\@#
    @ICCID
    WRITE ICCID
    @NAME
    SELECT ACCT
    SELECT NAME
    WRITE NAME
    @ACCTID
    SELECT ACCTID
    WRITE ACCTID
    @EXPIRE
    SELECT EXPIRE
    WRITE EXPIRE
$PR
```

As each card is issued, the personalization equipment characteristic data shown above is serially processed in four steps defined by the entries preceded by a "$." The card application record **922** is used to determine the value of the variable parameter fields in each instruction.

The $EMBOSS instruction is a single stream of data that begins with the control sequence #EMB# which notifies the personalization equipment that the data that follows should be embossed on the card. Each data field in the instruction is enclosed in a pair of percent signs. In this case, the first data field is FMTACCT, or the formatted account field as defined in the card application record **922**. The system **100** searches the card application record **922** for the FMTACCT entry and creates the string "1265-36830-91245" from the second data field in the first sample card holder record **952**. The next field, NAME, is taken from the first data field in the card holder record **952**. Thus, the emboss instruction for the first sample card holder record **952** becomes #EMB%1265-36830-91245%%Smith,James%.

The $ENCODE instruction causes the system **100** to process the card holder data to be encoded on the magnetic stripe of the card in the same fashion as the emboss instruction. Additional control characters in accordance with following IATA (International Air Travel Association) and ISO standards are inserted into the command. The resulting instruction is #ENC#%%%12653683091245%%Smith, James%.

The $IC command specifies the information to be stored in the chip's memory. The card operating system record **924** is used to translate the instructions in the personalization equipment record into the programming control commands for the operating system. A control sequence, #/@#, is used to notify the personalization equipment that the data that follows is chip data. The first field to be stored is the chip identifier, ICCID. The system **100** interprets the WRITE tag in the personalization equipment record **926** in accordance with the command identified with the WRITE tag in the card operating system record **924**. Since no offset value is specified in the application record **922** for the chip identifier entry, the default of "0000" is loaded into the %O variable parameter field. The %L variable parameter field is set to the value of the SIZE field in the $CHIP entry in the card framework template, i.e., "10" or hexadecimal "0A." The %D variable parameter field is set to the value of ICCID, "1234509876". The resulting command is A0D000000A1234509876.

The next commands cause the card operating system to store the card holder name into the account name file in the account directory on the chip. The system **100** translates the SELECT ACCT command into the corresponding card operating system command. The system **100** locates the SELECT entry in the card operating system record **924**, the ACCT entry in the card framework template record, and substitutes the specified directory path for the account directory defined in the ACCT entry, i.e. "5F10," for the %F variable parameter field in the command defined in the SELECT entry. The resulting command is A0A40000025F10. Similarly, the SELECT NAME command causes the system **100** to substitute the account name file "6F00" for the %F variable parameter field. The resulting command is A0A40000026F00. The final command in this series is the WRITE command. The system **100** interprets the WRITE command by substituting the default offset of "0000" for %O, the value of the SIZE field, "30" or hex "1E," as defined by the NAME entry in the card framework template record for %L, and the card holder's name, "Smith, James" for the first sample card holder data record **952**, for %D, to produce the command A0D000001ESmith, James~~~~~~~~~~~~~~~~~~~~~~ where each "~" represents a trailing space inserted to pad the name out to thirty characters.

The system **100** processes the remainder of the commands in the personalization equipment record **926** in a similar fashion to produce a contiguous string of data containing the commands to issue a card for the first sample card holder data record **952**:

#/@#A0D000000A123459876A0A40000025F10A0A40000026F00A0D000001ESmith, James~~~~~~~~~~~~~~~~~~~~~~A0A40000026F10A0A4000002E12653683091245A0A40000026F2040998.

The $PR command causes the system **100** to send the command data stream to the personalization equipment.

The data layouts shown in FIGS. **11**, **12** and **13**, and the sample data discussed in conjunction with the above example are only examples used to illustrate the functioning of various embodiments of the smart card personalization system **100**. That the layouts and data are necessarily defined by the environment in which they are used will be apparent to those skilled in the art.

As will also be apparent to those skilled in the art, the smart card personalization system **100** encompasses alternate embodiments of the software program in which the functions of the system are performed by modules different than those shown in the FIGS. The system **100** may process

the data in a serial or parallel fashion, or a combination of the two, without departing from the spirit or scope of the invention. The software program may be written in one of several widely available programming languages and the modules may be coded as subroutines, subsystems, or objects depending on the language chosen. Similarly, data used by the system **100** is described and represented as logical records embodied in a database but the invention is not limited to the described arrangement of data records, nor is the use of any particular type of data management system implied. Relational database systems from vendors such as Oracle, Sybase, Informix, or Microsoft provide the necessary infrastructure for managing the underlying data in the system, whether it is centralized or distributed, but other organizational data structures, i.e., indexed flat files, may be substituted without exceeding the scope of the invention.

Furthermore, alternate embodiments of the invention which implement the system in hardware, firmware, or a combination of both hardware and software, as well as distributing the modules and/or the data in a different fashion will be apparent to those skilled in the art and are also within the scope of the invention.

It is to be understood that the above description is intended to be illustrative, and not restrictive. Many other embodiments will be apparent to those of skill in the art upon reviewing the above description. The scope of the invention should, therefore, be determined with reference to the appended claims, along with the full scope of equivalents to which such claims are entitled.

What is claimed is:

1. A method for issuing portable programmed data carriers comprising:

acquiring a personalization equipment identifier and personalization data for a card holder from a card issuer management system;

acquiring equipment characteristic data for a personalization equipment type from a record in a database identified by the personalization equipment identifier; and

transferring the personalization data to personalization equipment as specified by the equipment characteristic data for the type of personalization equipment to issue the data carrier.

2. The method of claim **1**, further comprising translating the personalization data into an internal format such that the translated personalization data is transferred to the personalization equipment.

3. The method of claim **2**, wherein the personalization data is translated from a format defined by the card issuer management system into the internal format in accordance with format template data.

4. The method of claim **3**, further comprising acquiring the format template data from a record in the database identified by a data format identifier supplied by the card issuer management system.

5. The method of claim **3**, further comprising acquiring the format template data from the card issuer management system.

6. The method of claim **3**, further comprising acquiring the format template data from an application data record in the database identified by an application program identifier supplied by the card issuer management system.

7. The method of claim **1**, further comprising:

collecting information regarding the issuing of the data carriers; and

reporting statistics derived from the collected information to the card issuer management system.

8. The method of claim **1**, further comprising:

acquiring an application program identifier from the card issuer management system;

acquiring application data from a record in the database identified by the application program identifier; and

transferring the application data to the personalization equipment as specified by the equipment characteristic data.

9. The method of claim **1**, further comprising:

acquiring security data from a security source; and

transferring the security data to the personalization equipment as specified by the equipment characteristic data.

10. The method of claim **1**, further comprising:

acquiring a card operating system identifier from the card issuer management system;

acquiring programming control commands from a record in the database identified by the operating system identifier; and

transferring the programming control commands to the personalization equipment as specified by the equipment characteristic data.

11. The method of claim **10**, further comprising:

acquiring an application program identifier from the card issuer management system;

acquiring the application data from a record in the database identified by the application program identifier; and

transferring the application data to the personalization equipment as specified by the equipment characteristic data.

12. A system for issuing portable programmed data carriers comprising:

a card issuer management system interface for acquiring a personalization equipment identifier and personalization data for a card holder from a card issuer management system;

a personalization equipment interface for acquiring equipment characteristic data for a personalization equipment type from a record in a database identified by the personalization equipment identifier; and

the personalization equipment interface for further transferring the personalization data to personalization equipment as specified by the equipment characteristic data for the type of personalization equipment to issue the data carrier.

13. The system of claim **12**, wherein the system further acquires format template data from a record in a database identified by a data format identifier supplied by the card issuer management system and translates the personalization data into an internal format from a format defined by the format template data such that the personalization equipment interface transfers the translated personalization data to the personalization equipment.

14. The system of claim **12**, further comprising a tracking/report engine for collecting data from the system regarding the issuing of the data carriers and for reporting the collected data to the card issuer management system.

15. The system of claim **12**, further comprising:

a card application interface for acquiring application data from a record in the database identified by an application program identifier acquired by the card issuer management system interface; and

the personalization equipment interface for further transferring the application data to the personalization equipment as specified by the equipment characteristic data.

16. The system of claim 12, further comprising a security manager for acquiring security data from a security source and transferring the security data to the personalization equipment interface.

17. The system of claim 12, further comprising:

a card operating system interface for acquiring programming control commands from a record in a database identified by a card operating system identifier acquired by the card issuer management system interface; and

the personalization equipment interface for further transferring the programming control commands to the personalization equipment as specified by the equipment characteristic data.

18. The system of claim 17, further comprising:

a card application interface for acquiring application data from a record in the database identified by an application program identifier acquired by the card issuer management system interface; and

the personalization equipment interface for further transferring the application data to the personalization equipment as specified by the equipment characteristic data.

19. A data structure stored on a storage device for producing portable programmed data carriers comprising a plurality of personalization equipment elements, wherein each personalization equipment element is addressed by a unique personalization equipment identifier and specifies operating parameters for a type of personalization equipment such that the personalization data is properly formatted for transmission to the personalization equipment used to issue the data carrier.

20. The data structure of claim 19, further comprising a plurality of data format elements, wherein each data format element is addressed by a unique data format identifier and specifies a template used by a card issuer to format personalization data.

21. The data structure of claim 19, further comprising a plurality of card operating system elements, wherein each card operating system element is addressed by a unique card operating system identifier and specifies programming control commands for transmission to the personalization equipment.

22. The data structure of claim 19, further comprising a plurality of application program elements, wherein each application program element is addressed by a unique application program identifier and specifies application data used by a particular type of application program for transmission to the personalization equipment.

23. The data structure of claim 22, further comprising a plurality of card operating system elements, wherein each card operating system element is addressed by a unique card operating system identifier and specifics programming control commands for transmission to the personalization equipment.

24. A system for issuing portable programmed data carriers comprising:

personalization equipment receiving a data stream and in response thereto personalizing portable programmed data carriers;

personalization data obtained from a card issuer management system; and

a smart card personalization system having a database containing one or more data elements selected from the group consisting of
data format template elements,
card application data elements,
card operating system elements,
and personalization equipment elements,

wherein the smart card personalization system outputs the data stream as a result of processing the personalization data as directed by at least one of the selected data elements.

25. A method for issuing portable programmed data carriers comprising:

acquiring personalization data for a card holder and equipment characteristic data; and

transferring the personalization data to personalization equipment as specified by the equipment characteristic data to issue the data carrier.

26. A system for issuing portable programmed data carriers comprising a system interface for acquiring personalization data for a card holder and equipment characteristic data, and for further transferring the personalization data to personalization equipment as specified by the equipment characteristic data to issue the data carrier.

* * * * *

US005293424A

United States Patent [19]

Holtey et al.

[11] Patent Number: 5,293,424

[45] Date of Patent: Mar. 8, 1994

[54] SECURE MEMORY CARD

[75] Inventors: Thomas O. Holtey, Newton, Mass.; Peter J. Wilson, Leander, Tex.

[73] Assignee: Bull HN Information Systems Inc., Billerica, Mass.

[21] Appl. No.: 960,748

[22] Filed: Oct. 14, 1992

[51] Int. Cl.^{5} H04K 1/00; G06K 5/00; H04L 9/00

[52] U.S. Cl. 380/23; 235/380; 235/382; 380/24

[58] Field of Search 235/380, 382; 380/23, 380/24

[56] References Cited

U.S. PATENT DOCUMENTS

4,382,279	5/1983	Ugon	364/200
4,650,975	3/1987	Kitchener	235/375
4,742,215	5/1988	Daughters et al.	235/487
4,907,273	3/1990	Wiedemer	380/5
4,960,982	10/1990	Takahira	235/382
4,985,920	1/1991	Seki et al.	380/23
5,048,085	9/1991	Abraham et al.	380/23
5,148,481	9/1992	Abraham et al.	380/46

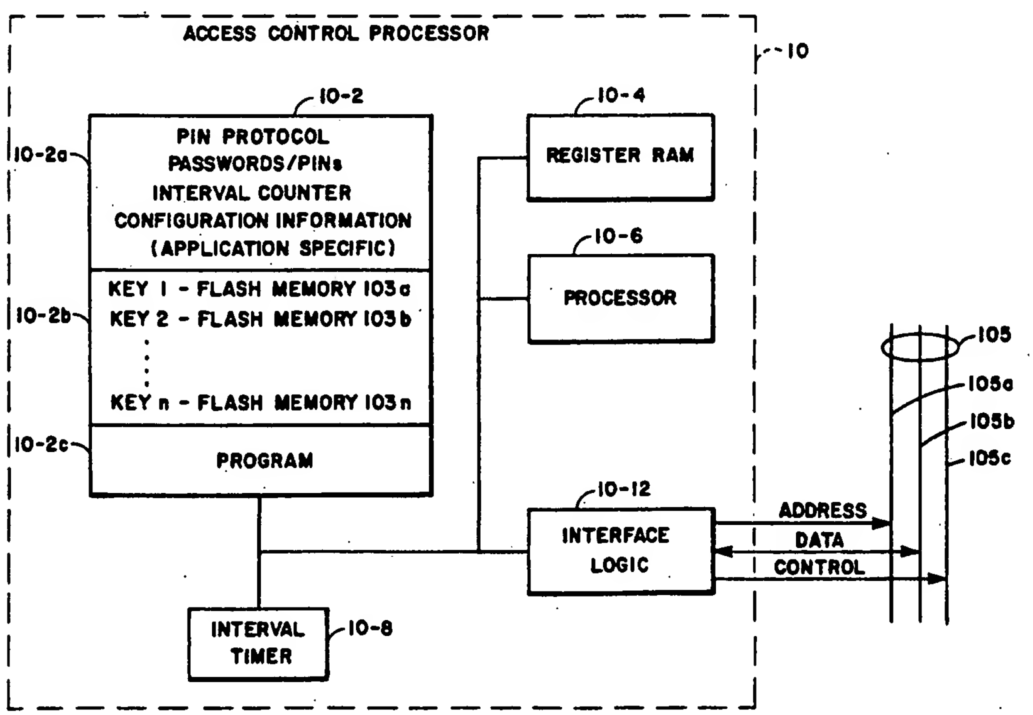
Primary Examiner—Stephen C. Buczinski

Attorney, Agent, or Firm—Faith F. Driscoll; John S. Solakian

[57] ABSTRACT

A secure memory card includes a microprocessor on a single semiconductor chip and one or more non-volatile addressable memory chips. The microprocessor chip and non-volatile memory chips connect in common to an internal card bus for transmitting address, data and control information to such non-volatile memory chips. The microprocessor includes an addressable non-volatile memory for storing information including a number of key values, application specific configuration information and program instruction information. Each chip's memory is organized into a number of blocks or banks and each memory chip is constructed to include security control logic circuits. These circuits include a number of non-volatile and volatile memory devices which are loaded with key and configuration information under the control of the microprocessor only after the microprocessor has determined that the user has successfully performed a predetermined authentication procedure with a host computer. Thereafter, the user is allowed to read out information from blocks only as defined by the configuration information.

26 Claims, 5 Drawing Sheets



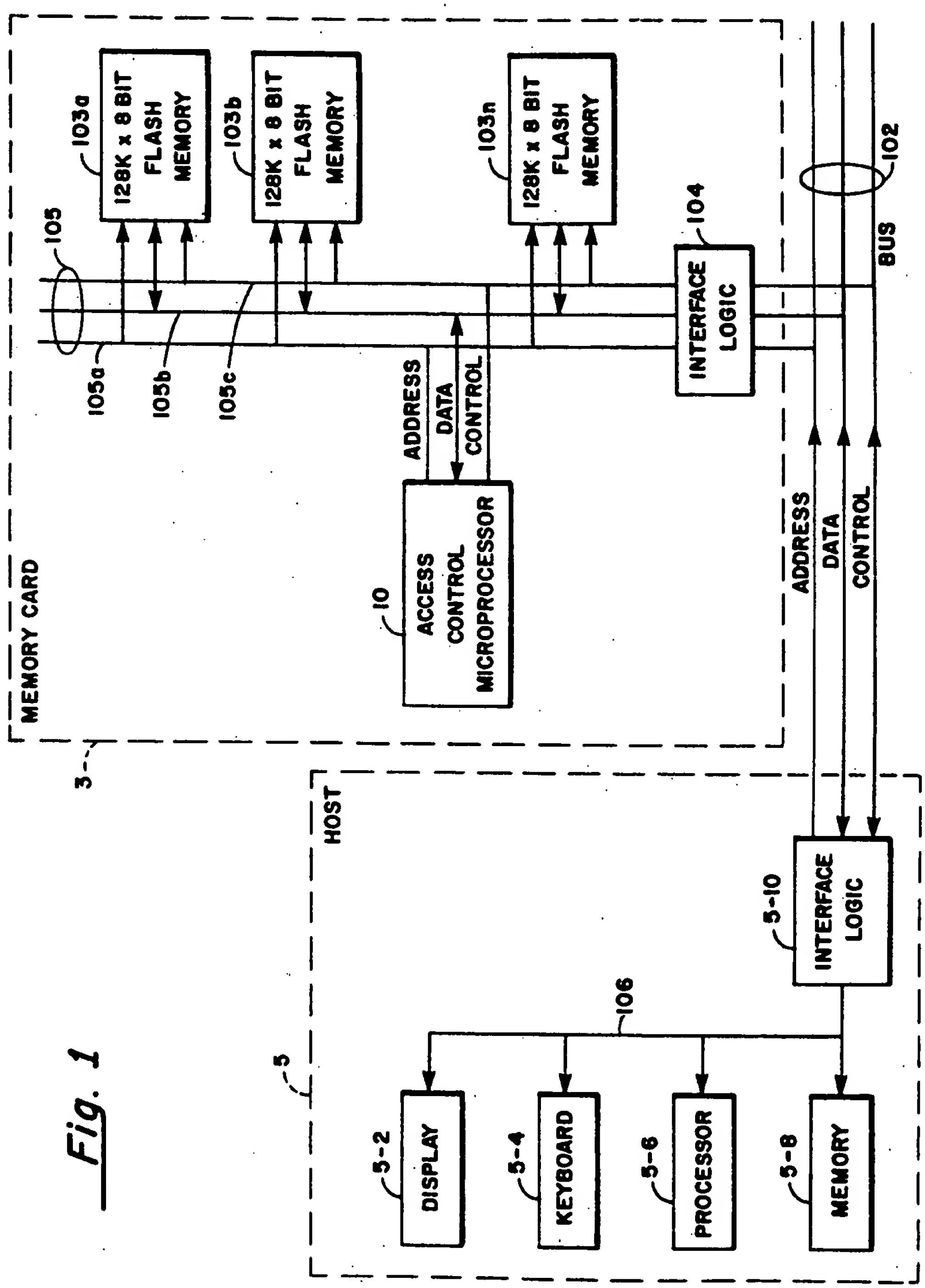
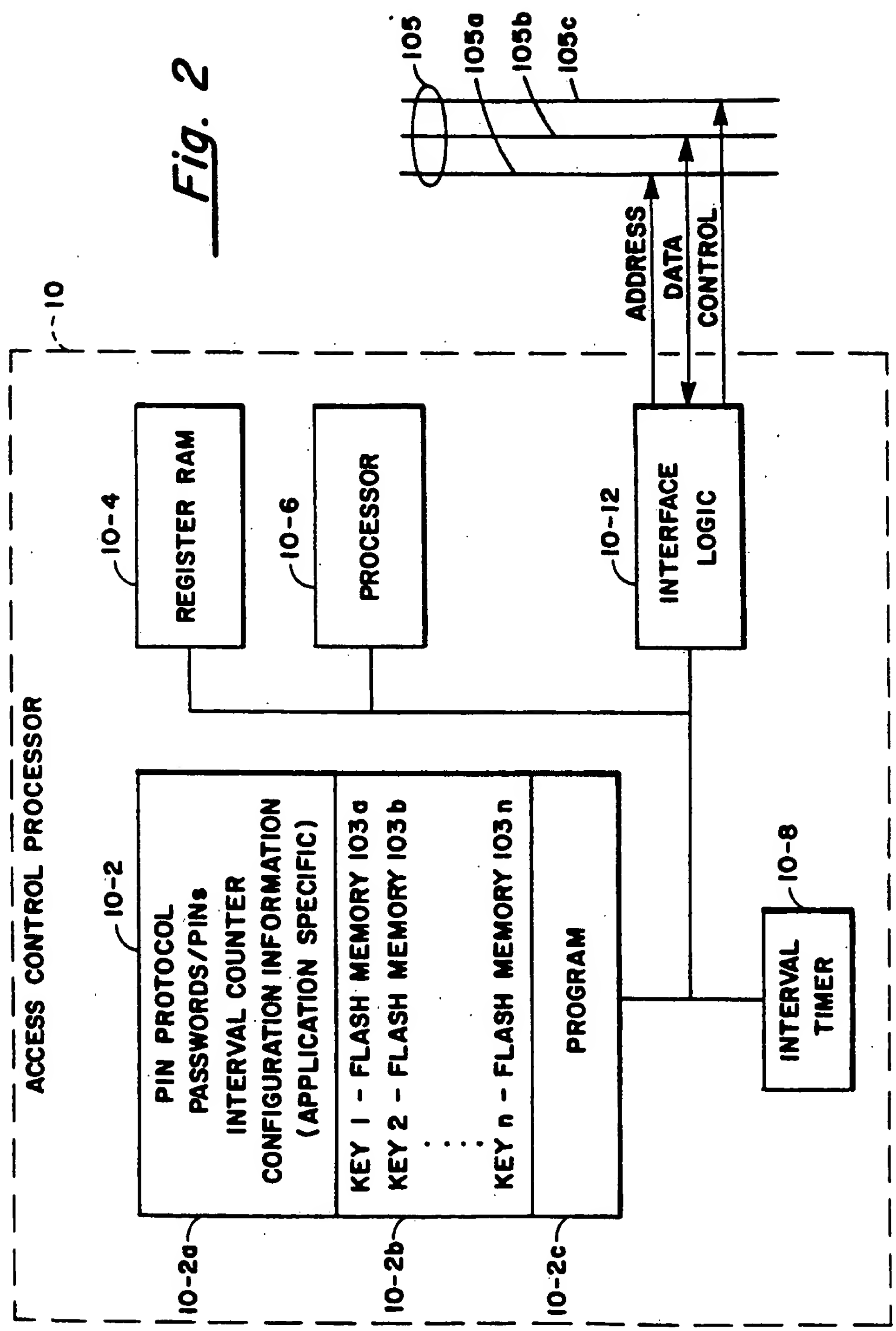

MEMORY CARD
128K x 8 BIT FLASH MEMORY
103a
103b
103n
105
105a
105b
105c
ADDRESS
DATA
CONTROL
10
ACCESS CONTROL MICROPROCESSOR
104
INTERFACE LOGIC
102
BUS
3
HOST
5
5-10
106
5-2
DISPLAY
5-4
KEYBOARD
5-6
PROCESSOR
5-8
MEMORY

Fig. 1



ACCESS CONTROL PROCESSOR
10
10-2
PIN PROTOCOL
PASSWORDS/PINs
INTERVAL COUNTER
CONFIGURATION INFORMATION
(APPLICATION SPECIFIC)
KEY 1 - FLASH MEMORY 103a
KEY 2 - FLASH MEMORY 103b
KEY n - FLASH MEMORY 103n
PROGRAM
10-2a
10-2b
10-2c
10-4
REGISTER RAM
10-6
PROCESSOR
10-12
INTERFACE LOGIC
10-8
INTERVAL TIMER
ADDRESS
DATA
CONTROL
105
105a
105b
105c

Fig. 2

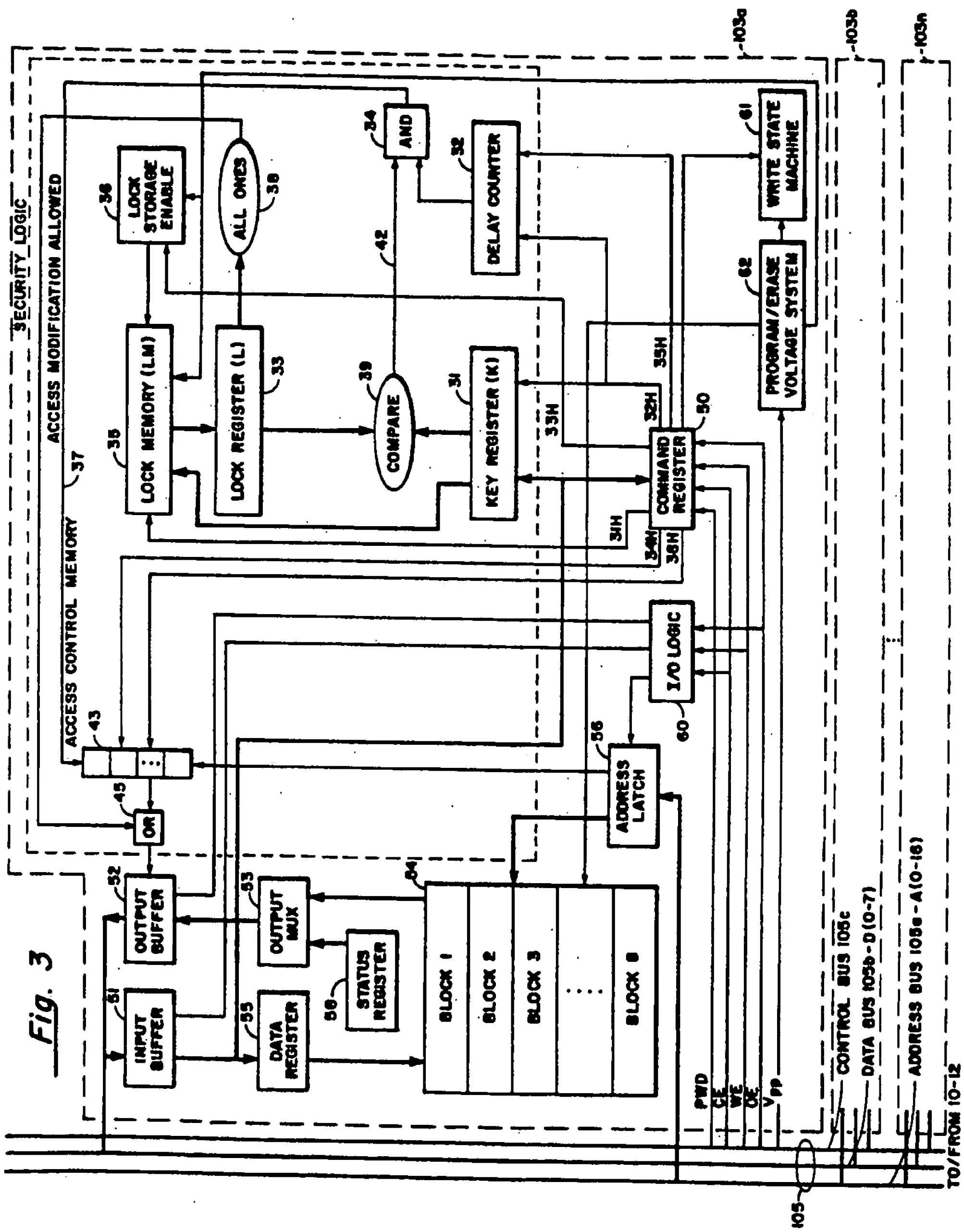
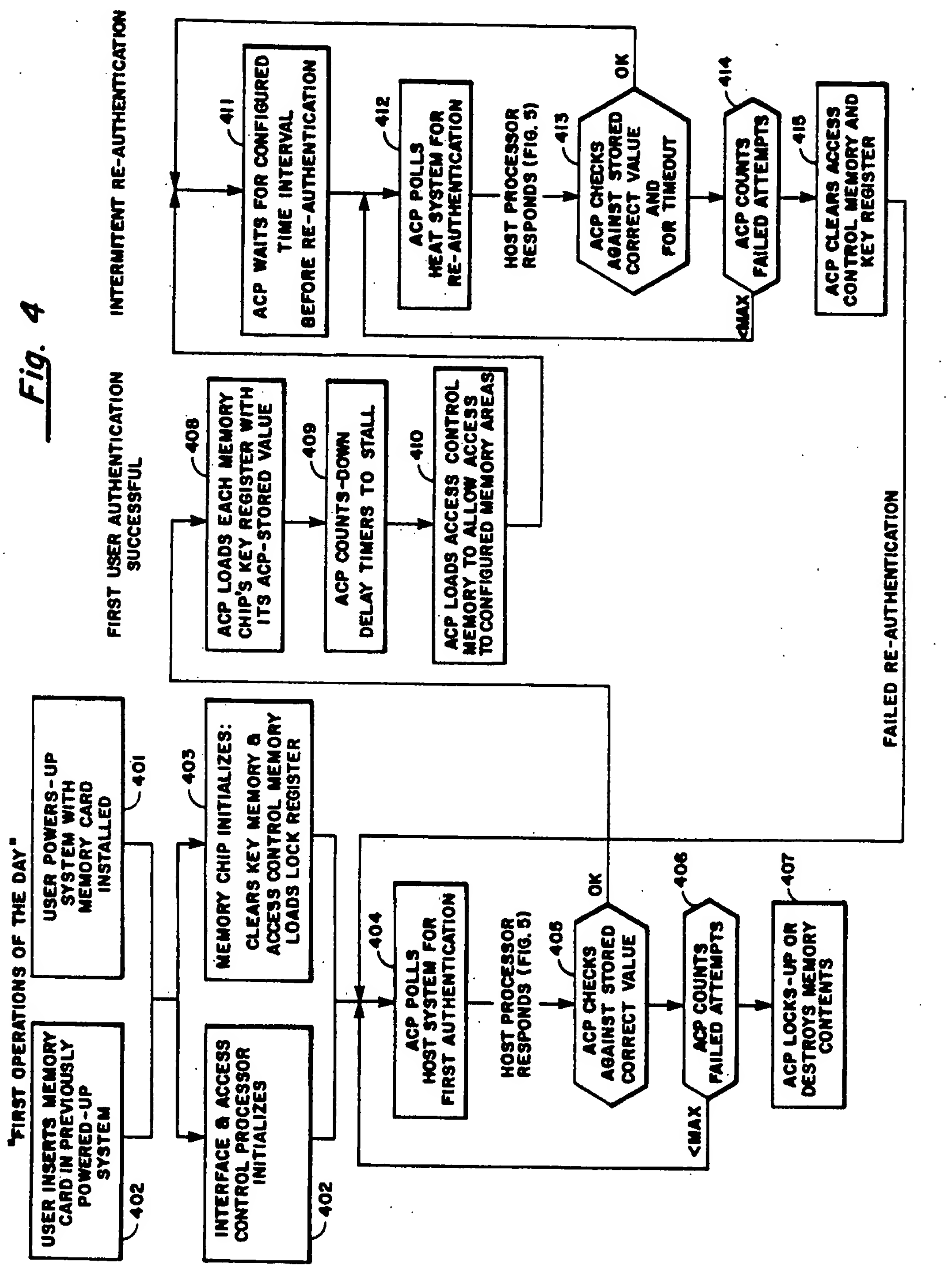

SECURITY LOGIC
ACCESS MODIFICATION ALLOWED
ACCESS CONTROL MEMORY
LOCK STORAGE ENABLE
ALL ONES
AND
DELAY COUNTER
WRITE STATE MACHINE
PROGRAM/ERASE VOLTAGE SYSTEM
LOCK MEMORY (LM)
LOCK REGISTER (L)
COMPARE
KEY REGISTER (K)
COMMAND REGISTER
I/O LOGIC
ADDRESS LATCH
OR
OUTPUT BUFFER
INPUT BUFFER
OUTPUT MUX
STATUS REGISTER
DATA REGISTER
BLOCK 1
BLOCK 2
BLOCK 3
BLOCK 8
PWD
CE
WE
OE
CONTROL BUS 105c
DATA BUS 105b-D(0-7)
ADDRESS BUS 105a-A(0-16)
TO/FROM 10-12
31H
32H
33H
34H
35H
38H
103a
103b
103n
105

Fig. 3


"FIRST OPERATIONS OF THE DAY"
USER INSERTS MEMORY CARD IN PREVIOUSLY POWERED-UP SYSTEM
402
USER POWERS-UP SYSTEM WITH MEMORY CARD INSTALLED
401
INTERFACE & ACCESS CONTROL PROCESSOR INITIALIZES
402
MEMORY CHIP INITIALIZES: CLEARS KEY MEMORY & ACCESS CONTROL MEMORY LOADS LOCK REGISTER
403
ACP POLLS HOST SYSTEM FOR FIRST AUTHENTICATION
404
HOST PROCESSOR RESPONDS (FIG. 5)
ACP CHECKS AGAINST STORED CORRECT VALUE
405
OK
ACP COUNTS FAILED ATTEMPTS
406
<MAX
ACP LOCKS-UP OR DESTROYS MEMORY CONTENTS
407
FIRST USER AUTHENTICATION SUCCESSFUL
ACP LOADS EACH MEMORY CHIP'S KEY REGISTER WITH ITS ACP-STORED VALUE
408
ACP COUNTS-DOWN DELAY TIMERS TO STALL
409
ACP LOADS ACCESS CONTROL MEMORY TO ALLOW ACCESS TO CONFIGURED MEMORY AREAS
410
INTERMITENT RE-AUTHENTICATION
ACP WAITS FOR CONFIGURED TIME INTERVAL BEFORE RE-AUTHENTICATION
411
ACP POLLS HEAT SYSTEM FOR RE-AUTHENTICATION
412
HOST PROCESSOR RESPONDS (FIG. 5)
ACP CHECKS AGAINST STORED CORRECT VALUE AND FOR TIMEOUT
413
OK
ACP COUNTS FAILED ATTEMPTS
414
<MAX
ACP CLEARS ACCESS CONTROL MEMORY AND KEY REGISTER
415
FAILED RE-AUTHENTICATION
Fig. 4

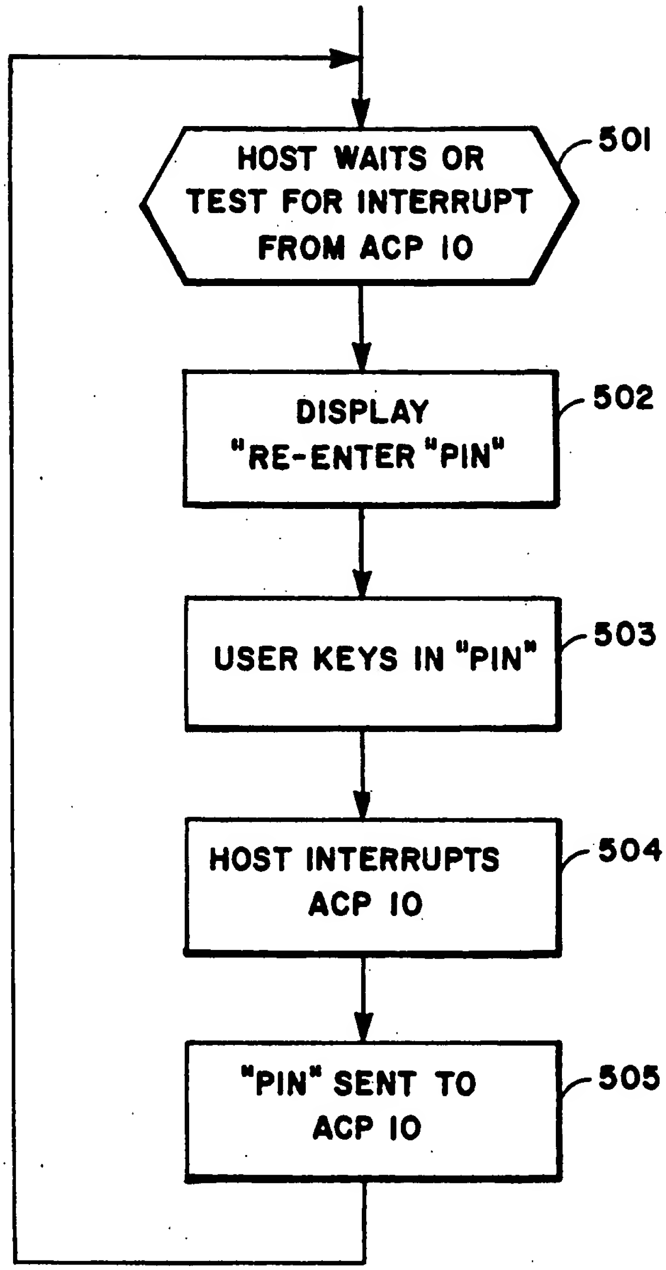

HOST WAITS OR TEST FOR INTERRUPT FROM ACP 10
501
DISPLAY "RE-ENTER "PIN"
502
USER KEYS IN "PIN"
503
HOST INTERRUPTS ACP 10
504
"PIN" SENT TO ACP 10
505

Fig. 5

SECURE MEMORY CARD

BACKGROUND OF THE INVENTION

1. Technical Field

This invention relates to the field of portable personal computers and more particularly to maintaining systems for data security in a portable digital information environment.

2. Description of the Prior Art

The security of personal information has forever been a concern. It has been ensured by locks, codes and secret pockets. As information has taken new forms, new methods have been required to meet the changed situations.

Historically, security of information has been addressed by use of signatures, credentials and photographs. Electronic devices such as automatic banking machines have added encoded cards and personal identification numbers (PINS) to the repertoire of security tools. Computer systems continue to use passwords.

More recently the "Smart Card" has been used as a security tool. The "Smart Card" is a small microcomputer with writable, non-volatile memory and a simple input/output interface, fabricated as a single chip and embedded in a plastic "credit card". It has exterior pads to allow it to be connected to specially designed equipment. The program contained in the card's microcomputer interacts with this equipment and allows its non-volatile memory data to be read or modified according to the desired algorithm which may optionally include a password exchange. Special techniques have been implemented to protect the memory information and to allow varied permissions according to the situation. For example, U.S. Pat. No. 4,382,279 entitled, "Single Chip Microprocessor with On-Chip Modifiable Memory" discloses an architecture which permits automatic programming of a non-volatile memory which is included on the same chip as a processing and control unit. As in other systems, the microprocessor only protects memory on the same chip.

The "Smart Card" has been used both to facilitate the process of identification and to be the actual site of the valued information. In this situation, as in most past situations, physical presence of a "key" as well as some special knowledge has been used as part of the verification or authentication process. In such above cases, identification has been a dialog between the person desiring access and a fixed agency such as a security guard or an automatic teller machine.

The current state of portability of freestanding computing devices makes it possible for both the physical key and the authentication agent to be small, portable and hence more subject to loss or theft. Further, computing devices make it possible to perform repeated attempts to guess or deduce the special knowledge or password associated with the identification process. This is especially true if the authentication agent or device is also in the control of the thief or burglar. To make matters worse, technology now allows and encourages the carrying of enormous amounts of sensitive information in a pocket or handbag where it is subject to mishap.

Today, notebook and subnotebook sized computers provide a capable freestanding environment which allows for significant computing power and thus creates a need for additional data storage capability. This has initially been met by miniature hard disk devices which hold both programs and data. While password protection is often used in these systems, it does not completely protect sensitive data because, first, the authentication agent is itself vulnerable. However, more significantly, the disk drive containing the data can be physically removed and accessed in a setting more conducive to data analysis. In this case, only some form of encryption is capable of protecting the data. The nature of disk access makes this possible without undue performance or cost barriers. An example of this type of system is described in U.S. Pat. No. 4,985,920 entitled, "Integrated Circuit Card."

The recent emergence of the flash memory and removable "memory cards" has allowed major reductions in size and power requirements of the portable computer. The flash memory combines the flexibility of random access memory (RAM) with the permanence of disks. Today, the coupling of these technologies allows up to 20 million bytes of data to be contained, without need of power, in a credit card size, removable package. This data can be made to appear to a host system either as if it were contained in a conventional disk drive or as if it were an extension of the host's memory. These technological developments have made further reduction in system size possible to the extent that it may be carried in a pocket rather than in a handbag or briefcase.

Thus, the data and its host system have become more vulnerable to loss or theft and simultaneously more difficult to protect memory data by encryption as this presents major cost and performance barriers.

Accordingly, it is a primary object of the invention to provide a portable digital system with a secure memory subsystem.

It is another object of the invention to provide a memory card which can be protected if removed from a portable digital system.

It is still a further object of the present invention to provide a memory card in which the chips of the card are protected if removed from such card.

SUMMARY OF THE INVENTION

The above objects are achieved in the secure card of a preferred embodiment of the present invention. The secure memory card includes a microprocessor on a single semiconductor chip and one or more non-volatile addressable memory chips. The microprocessor chip and nonvolatile memory chips connect in common to an internal card bus for transmitting address, data and control information to such non-volatile memory chips. The microprocessor includes an addressable non-volatile memory for storing information including a number of key values, configuration information and program instruction information for controlling the transfer of address, data and control information on the internal bus. The chip memory is organized into a number of blocks or banks, each block having a plurality of addressable locations.

According to the present invention, each memory chip is constructed to include security control logic circuits. In the preferred embodiment, these circuits include a non-volatile lock memory, a non-volatile lock storage enable element and a volatile access control memory, each being loadable under the control of the microprocessor. More specifically, the microprocessor first loads a lock value into the non-volatile lock memory and resets the lock storage enable element inhibiting access. Thereafter, the microprocessor loads the access

control memory as specified by the configuration information. Such information is loaded only after the microprocessor has determined that the user has successfully performed a predetermined authentication procedure with a host computer. The security logic circuits of each memory enable the reading of information stored in selected addressed blocks of the flash memory as a function of the configuration information loaded into the memory chip's access control memory. Periodically, the user is required to successfully perform an authentication procedure with the host computer, and the user is allowed to continue reading information as allowed by the access control memory. In the preferred embodiment, the host computer is coupled to the memory card through a standard interface such as the interface which conforms to the Personal Computer Memory Card International Association (PCMCIA) standards.

The present invention melds the "SmartCard" and "memory card" technologies which is key to allowing the protection of the large amounts of data made possible by the flash memory technology in the "security harsh" environments which electronic miniaturization has created. Further, the present invention is able to take advantage of improvements and enhancements in both technologies.

Additionally, the security logic circuits of the present invention are incorporated into and operate in conjunction with the flash memory in a way that minimizes the amount of changes required to be made to the basic logic circuits of the flash memory. More specifically, the flash memory can be operated in a secure mode and in a non-secure mode wherein the security logic circuits are bypassed enabling the flash memory to operate as if such circuits had not been installed. The non-secure mode is normally entered when the contents of the flash memory's non-volatile lock memory are cleared. This is generally indicative of an unprogrammed or fully erased flash memory which naturally erases to a predetermined state (i.e. an all ONES state).

With the addition of a small amount of logic to the flash memory and an "Access Control Processor" (ACP), the contents of the flash memory is made secure without requiring data encryption. Therefore, the invention eliminates the overhead of encrypting and decrypting data which can be quite time-consuming for large blocks of data.

In operation, the ACP periodically prompts the user of the system for entry of some form of authentication. This may be a password, a PIN, a specific pen computer "gesture" performed at a specific point on the writing surface, a spoken command or a "voiceprint" of the user. The method varies with the system. The programmable ACP allows the user to alter the specific content of the authentication and the frequency of prompting. The code for authentication and the data required by the lock and access control memories are stored within the ACP's non-volatile memory which is on the same chip as the ACP and, hence, are protected.

As mentioned, a successful authentification causes the ACP to enable, or continue to enable, all or selected blocks of the flash memory for access. Failure causes access to the flash memory to be disabled. Thus, the operation is similar to a "dead man throttle" in that any failure to successfully complete authentication will cause the flash memory's data to be protected. In addition, a command initiated by the user can also cause access to be disabled. Further, upon first application of power from a powered off condition, access is blocked to protected memory contents until the first authentication is successfully performed.

Thus, if either the memory card or its host processor is lost, stolen, powered off or left unattended, the memory's data is protected from access, either immediately or as soon as the current periodic authentication expires. In the event of theft, the memory data is protected from access even if the memory card is opened and probed electronically or the memory chips are removed and placed in another device.

The above objects and advantages of the present invention will be better understood from the following description when taken in conjunction with the accompanying drawings.

BRIEF DESCRIPTION OF THE DRAWING

FIG. 1 shows an overall block diagram of a system which incorporates the memory card constructed according to the present invention.

FIG. 2 shows in greater detail, the access control processor (ACP) of FIG. 1 including a layout of its non-volatile memory.

FIG. 3 shows a detailed block diagram of a standard flash memory of FIG. 1 modified according to the present invention.

FIGS. 4 and 5 are flow charts used to explain the operation of the memory card of the present invention in carrying out various authentication procedures.

DESCRIPTION OF THE PREFERRED EMBODIMENT

FIG. 1 is a block diagram of a secure portable handheld computing system 1 usable as a personal computer or as a transaction processor. System 1 includes a memory card 3 constructed according to the present invention which connects to a host processor 5 by a bus 102. The host processor 5 may take the form of a palm top personal computer, such as the HP 95LX manufactured by Hewlett-Packard. The host processor 5 includes a liquid crystal display (LCD) 5-2, a keyboard 5-4, a microprocessor 5-6, a memory 5-8 and a serial interface 5-10 all coupled in common to a bus 106. The memory 5-8 includes a one megabyte read only memory (ROM) and a 512 Kbyte random access memory (RAM).

The connection between the memory card 3 and host processor 5 is established through a standard bus interface. In the preferred embodiment, the bus 102 conforms to the Personal Computer Memory Card International Association (PCMCIA) standard. The interface 102 provides a path for transferring address, control and data information between host processor 5 and the memory card system 3 via a standard interface chip 104 and a memory card bus 105. Each of the buses 102, 105 and 106 include a data bus, a control bus and an address bus and provide continuous signal paths through all like buses. For example, bus 105 includes address bus 105*a*, data bus 105*b*, and control bus 105*c*.

The PCMCIA bus standard has evolved from a standard which supports disk emulation on memory cards to a substantially different standard which allows random access to memory data. The memory card of the present invention provides a protection technique which supports this new standard by providing rapid access to random memory locations without resort to encryption techniques. By controlling the data paths which carry the data from the memory array to the host, the memory card of the present invention protects

the data without imposing any time-consuming buffering, decryption or other serial processing in this path.

Typically, a user operates system 1 from the keyboard 5-4 to perform the typical operations such as spreadsheet and database functions which display information on display 5-2 and update information stored in files in memory card 3. The host processor 5 sends address information over bus 102 to retrieve information and if desired, updates the information and sends it, along with the necessary address and control information back to memory card 3.

As shown in FIG. 1, the memory card 3 of the present invention includes an access control processor (ACP) 10 coupled to bus 105 and a number (n) of CMOS flash memory chips 103*a* through 103*n*, each coupled to bus 105. ACP 10 is typically the same type of processing element as used in the "Smart Card". The CMOS flash memories 103*a* through 103*n* may take the form of flash memory chips manufactured by Intel Corporation. For example, they make take the form of the Intel flash memory chip designated as Intel 28F001BX 1M which includes eight 128 KBYTE×8 CMOS flash memories. Thus, a 4-MBYTE flash memory card could include 32 CMOS flash memories, that is 'n'=32.

ACCESS CONTROL PROCESSOR 10

FIG. 2 shows in block diagram form, the access control processor (ACP) 10 of the preferred embodiment. As shown, ACP 10 includes a protected non-volatile memory 10-2, a random access memory (RAM) 10-4, a microprocessor 10-6, an interval counter 10-8 and an interface block 10-10 connected to bus 105. Non-volatile memory 10-2 dedicates a number of addressed locations in which to store authentication information and programs. More specifically, memory locations 10-2*a* store one or more personal identification numbers (PINs), protocol sequences or other identification information for verifying that the user has access to the system, and for identifying the blocks in flash memories 103*a* through 103*n* that the user may access in addition to a time interval value used for reauthentication.

Memory locations 10-2*b* store the key values used for protecting each of the flash memories 103*a* through 103*n* or the codes used to protect the individual blocks of each of the flash memories 103*a* through 103*n*.

Memory locations 10-2*c* store the program instruction sequences for performing the required authentication operations and for clearing the system if the preset conditions for failure are met. Certain program instructions enable the user to control the setting of the interval counter 10-8 which establishes when user re-authentication takes place. The reauthentication interval defines the time between interruptions and for sending an interrupt to the host processor 5 requiring verification of the user's identity by having the user reenter the PIN or other password. The interval counter 10-8 receives clock pulses from the host processor 5 over bus 102 and can be set by the user according to the work environment. For example, at home, the user may turn the timer off (i.e., set it to a maximum value), or set the time interval to one hour. On an airplane the user may set it for ten minutes for increased protection. As described herein, the user is prompted to re-examine the setting of this interval at every "power on" thereby forcing periodic re-authentications to enforce security.

FLASH MEMORIES 103*a* through 103*n*

FIG. 3 is a detailed block diagram of flash memories 103*a* through 103*n*. Only the detailed logic circuits of memory 103*a* are shown since memories 103*b* through 103*n* are constructed identically to memory 103*a*.

The flash memory 103*a* basically comprises two sections, a section containing the security access control circuits of the present invention and another section containing the basic or standard logic circuits of the flash memory.

Security Access Control Section

As seen from FIG. 3, the security control circuits of the present invention include a 32-bit key register, a 32-bit volatile lock register 33, a 12-bit delay counter 32, a comparator circuit 39, an all ONES detected signal circuit 38, a non-volatile lock memory 35, a one-bit non-volatile lock storage enable element 36, a volatile access control memory 43, an access modification allow AND gate 34 and an output OR gate 45 arranged as shown. It will be noted that this section receives command control signals designated by various hexadecimal values (e.g. 31H through 38H) from a command register 50 included in the basic logic section. These signals indicate the different data values of the set of commands received by the command register 50 from the ACP 10 via data bus 105*b*. These commands are an important extension to the sets of commands normally used by the flash memory. The standard flash memory commands take the form of the commands utilized by the 28F001BX flash memory. Those commands are described in the publication entitled, "Memory Products," published by Intel Corporation, referenced herein. The commands used by the present invention are described in Table 1.

Referring to Table 1, the first command shown is a load lock memory command which is used to initially load a random number generated lock value into non-volatile lock memory (LM) 35 in each memory 103*a* through 103*n*. Each memory 103*a* through 103*n* may have a different lock value or the same lock value depending on the security needs of the users. The lock value is loaded into LM 35 through key (K) register 31 under control of the one bit, non-volatile storage element 36. The reset lock storage enable command of Table 1 is used to reset storage element 36. This prevents the lock value stored in LM 35 from being changed since storage element 36 once reset by the reset lock storage enable command cannot be set. The non-volatile contents of LM 35 are transferred to the L register 33 on power-up. It will be noted that the location or site of lock memory 35 is design dependent. For example, memory 35 could be implemented as an extension to memory array 54.

The load key register command of Table 1 is used to load the key register 31 and set the delay counter 32. The decrement delay counter command is used by the ACP 10 to decrement by one, the contents of the delay counter 32. The read allow memory bank and read disable memory bank commands are used by the ACP 10 to enable or disable access to the different memory blocks of memory array 54 during loading of the access control memory 43.

TABLE 1

Command	First Bus Cycle Operation	Address	Data	Second Bus Cycle Operation Address	Data
Load Lock Memory	Write		31H	Write	N/A
Reset Lock Storage Enable	Write		33H	N/A	N/A
Load Key Register	Write		32H	Write	Key Data
Decrement Delay Counter	Write		35H	N/A	N/A
Read-Allow Memory Bank	Write	MBA	34H	Write	MBA
Read-Disable Memory Bank	Write	MBA	38H	Write	MBA

Load Lock Memory (31H)
This command copies the contents of the key register 31 into the non-volatile lock memory 35 if and only if the lock storage enable 36 output signal is TRUE.
Reset Lock Storage Enable (33H)
This command resets the lock storage enable logic element 36, thus inhibiting loading or changing the lock storage memory 35.
Load Key Register (32H)
This command shifts the prior contents of the key register 31, one byte (LSB toward MSB) and loads "Key Value" from ACP 10 into the key register LSB. Further, it sets the Delay Counter 32 to its maximum value, e.g., all ONES.
Decrement Delay Counter (35H)
This command decrements the delay counter 32 by ONE. The delay counter must equal ZERO to allow subsequent reading of the memory array 54.
Read-Allow Memory Bank (34H)
This command sets the bit corresponding to the memory bank address (MBA) in the access control memory 43 if and only if the access modification allowed signal 37 is TRUE. This allows read access to the selected bank.
Read-Disable Memory Bank (38H)
This command resets the bit corresponding to the memory bank address in the access control memory 43.

Considering Table 1 in greater detail, it is seen that Table 1 also shows the bus cycle operations for each of the added commands. For each command requiring two bus cycles, during each first bus cycle, the command register **50** receives an 8-bit command generated by ACP **10**, sent via the data bus **105***a* of bus **105** and an input buffer **51**. Command register **50** conditions the selected logic element to receive from data bus **105***b*, the information required to execute the command during a second bus cycle. As indicated, the second bus cycle is designated not applicable (N/A) since the reset lock storage enable and decrement delay counter commands need only one cycle for execution.

During normal operation, the K register **31** is loaded with the key value received from memory locations **10-2***b* by a load key register command and delay counter **32** is set to its maximum value. Delay counter **32** is decremented to all ZEROS in response to successive decrement delay counter commands received from the ACP **10** and generates a zero count output signal **41** which is applied as an input to AND **34**.

Each delay counter **32** limits the number of tries or attempts which can be made to access the flash memories **103***a* through **103***n* in the case where a thief removes the chips and places them upon the "outlaw card" and programs a processor or equipment to repeatedly try to guess each memory chip's key. Stated differently, counter **32** ensures that a significant number of tries or attempts must be made in order to gain illegal access to the flash memories. The key and delay counter sizes are selected to require such testing to take an unreasonable amount of time.

More specifically, the Key Register **31** stores approximately 4 billion (2^{32}) different combinations. In the preferred embodiment, the delay counter **32** is a twelve-bit counter. Assuming the delay counter **32** is decremented once each microsecond, it will require 2^{12} or 4 milliseconds per attempt at guessing the key value. The ACP **10**, knowing the correct key value, incurs only a four millisecond delay in the initial setup. Random attempts to guess the key value will require 2^{31} tries for a 50% chance of success. This would require 231×212 microseconds or 102 days to guess the key value. This time is sufficient to deter most thieves. Of course, a longer or shorter time could be provided by modifying the sizes of the key and delay counter **32**.

In the case where the memory card of the present invention is stolen and is put into an "outlaw host," the ACP **10** limits the number of tries by the thief to guess the PIN by known techniques. Such techniques may include locking access or destroying data if a threshold of incorrect guesses is exceeded.

During an initial authentication operation for flash memory **103***a*, a key value is loaded into the 32 bit K register **31** in response to four successive load key register commands (i.e., data bus **105***b* is a byte wide bus). Delay counter **32** is forced to its maximum count of (ALL ONE'S) and decremented by the ACP **10** sending decrement delay counter commands on successive first bus cycles. When the delay counter **32** is decremented to ZERO, it generates the zero count signal **41** which is applied to one input of AND gate **34**.

If the key value stored in the K register **31** equals the lock value stored in the corresponding L register **33** indicating that the user provided the proper identification to the host processor **5**, then compare logic **39** applies an equals compare signal **42** to another input of AND gate **34**. This causes AND gate **34** to generate an access modification allowed signal **37** at its output, which enables -writing to access control memory **43**, under the control of ACP **10**. This, in turn, subsequently allows the reading of memory array **54**.

The access control memory **43** contains volatile storage of one bit for each block/bank of the memory array **54**. These bits are cleared to ZERO as part of the flash memory's power up sequence. In order for data to be read from the memory **103***a*, the bit corresponding to the addressed memory block must be at logical ONE. These bits are set by the ACP **10** issuing read-allow memory bank commands if and only if the access modification allowed signal **37** is TRUE.

As shown in Table 1, during the second bus cycle of the read-allow memory bank command, the three (3) high order address bits of the selected memory bank of memory array 54 are sent over address bus 105c as well as a repeat of the hexadecimal command identifier being sent over the data bus 105a to command register 50. This results in a ONE being written into the addressed bit location in access control memory 43. In the preferred embodiment, the read-allow memory bank command sequence is repeated eight times since the memory array 54 is organized into eight banks of 16K bytes each. The ACP 10 may restrict access to selected banks by issuing a sequence of read-disable memory bank commands in a similar manner.

The output of the access control memory 43 of the present invention is applied as an enabling input to output buffer 52 during each flash memory read cycle when the contents of a location of any bank of memory array 54 is being read out. That is, a read cycle may occur, however, the data read out is inhibited from passing through output buffer 52 in the absence of the appropriate bank's access control memory gating signal. More specifically, in the case of the preferred embodiment, access control memory 43 includes eight individually addressable bit storage elements, an input address 3 to 8-bit decoder connected to the input of each storage element and a 1 to 8 output multiplexer circuit connected to the output of each storage element. The three high order address bits of each address are decoded and used to select the storage element for the block whose contents are to be changed. Similarly, the same three bits are used to select the output of the storage element for the block containing the flash memory location being read.

If the lock memory 35 is fully erased, i.e., at ALL ONES as indicated by the contents of the L register 33 being at all ONES, then the output buffer 52 is always enabled. That is, when lock register 33 contains "ALL ONES," this generates a signal from ALL ONES detector element 38 to the OR gate 45 to enable the output buffer 52. This effectively places flash memory 103a in non-secure mode. This allows all of the security logic circuits of the present invention to be bypassed. Hence, the same flash memory chip can be used for both secure and non-secure applications, thus resulting in production economies.

Flash Memory Basic Logic Circuits

As shown in FIG. 3, such circuits include a memory array 54, a command register 50, input/output logic circuits 60, an address latch 56, a write state machine 61, erase voltage system 62, an output multiplexer 53, a data register 55, input buffer 51, output buffer 52 and a status register 58, as shown. The basic logic circuits of flash memory 103a as discussed above, takes the form of the type of circuits included in the flash memory designated as 28F001BX manufactured by Intel Corporation. Since such circuits are conventional, they will only be described to the extent necessary. For further information regarding such circuits, reference may be made to pages 3-109 through 3-134 of the publication entitled, "Memory Products," order Number 210830, published by Intel Corporation, dated 1992. As shown in FIG. 3, the flash memory basic circuits receive a number of input signals (A0-A16), address, data signals (D00-D07) and control signals (CE, WE, OE, PWD and VPP). These signals are described below in Table 2.

TABLE 2

Symbol	Signal Descriptions Name and Function
A0-A16	ADDRESS INPUTS for memory addresses. Addresses are internally latched during a write cycle.
D00-D07	DATA INPUTS/OUTPUTS: Inputs data and commands during memory write cycles; outputs data during memory and status read cycles. The data pins are active high and float to tri-state off when the chip is deselected or the outputs are disabled. Data is internally latched during a write cycle.
CE	CHIP ENABLE: Activates the device's control logic, input buffers, decoders and sense amplifiers. CE is active low, CE high deselects the memory device and reduces power consumption to standby levels.
PWD	POWERDOWN: Puts the device in deep powerdown mode. PWD s active low; PWD high gates normal operation. PWD = VHH allows programming of the memory blocks. PWD also locks out erase or write operations when active low, providing data protection during power transitions.
OE	OUTPUT ENABLE: Gates the device's outputs through the data buffers during a read cycle. OE is active low.
WE	WRITE ENABLE. Controls writes to the command register and array blocks. WE is active low. Addresses and data are latched on the rising edge of the WE pulse.
Vpp	ERASE/PROGRAM POWER SUPPLY for erasing blocks of the array or programming bytes of each block. Note: With Vpp < Vppl Max, memory contents cannot be altered.

As shown in Table 2, the Chip Enable (CE), Write Enable processor (WE) and Output Enable (OE)) signals are applied to command register 50 and I/O logic 60 from host processor 5, via bus 102 and control bus 105b and are dispersed to control specified logic blocks. A powerdown (PWD) signal is also applied to command register 50 for enabling the flash memory to per-

form the operations specified in Table 2. This signal can be used to clear the volatile storage elements of the flash memory's security control section as desired thereby enforcing user reauthentication when normal operation is again resumed.

Generally, the basic logic elements of the flash memory operate in the following manner. Information is stored in memory array 54 via data bus 105a, input buffer 51 and data register 55 at an addressed location of one of the memory blocks specified by the address received by an address logic 56 from address bus 105c. Information is read from a specified address location of a bank of memory array 54 and is sent to host processor 5 via an output multiplexer 53, output buffer 52, data bus 105a and bus 102. Status register 58 is used for storing the status of the write state machine, the error suspend status, the erase status, the program status and the Vpp status.

The write state machine 61 controls the block erase and controls program algorithms. The program/erase voltage system 62 is used for erasing blocks of the memory array 54 or the programming bytes of each block as a function of the level of Vpp (i.e., when Vpp is at a high level programming can take place; if Vpp is at a low level, memory array 54 functions as a read only memory).

DESCRIPTION OF OPERATION

The operation of the secure memory card of the present invention will now be described with particular reference to the flow diagram of FIGS. 4 and 5. Before describing such operations in detail, the steps involved in the fabrication, customization and operation of the memory card will first be described.

As a first step, at card fabrication, the ACP 10 sets the lock value for each of the memory chips on the memory card. It does this by loading the key value into the lock memory of FIG. 3. These values are stored in the ACP's protected non-volatile memory 10-2 (i.e., keys 1-n in FIG. 2). The lock storage enable elements 36 are then set to ZEROs to inhibit further changing or reading of lock memory contents. As these elements are nonvolatile, they cannot be changed unless the entire flash memory chip is cleared.

As a second step, at application customization, since writing is not affected by the protection functionality, the memory card can then be loaded with its data or software application. The ACP 10 is then loaded with information pertaining to the memory's bank structure and the degrees of protection which are to be applied to each memory bank.

As a third step, at user customization, the user establishes parameters for the frequency and mode of authentication and specific data required (e.g., personal identification numbers (PINs)). This information is stored in the ACP'S memory.

As a fourth step, at power on, the "key register", "access modification allowed" signal and "access control memory" are initialized so as to inhibit access to data or writing to access control memory 43. The first authentication dialog is initiated.

At first authentication dialog, the ACP 10, using the services of its host processor 5, prompts the user and receives authentication information. If authentication is unsuccessful, no operation is performed; if successful, the key register of each memory chip is loaded with the value stored in the ACP'S memory. During this operation, the delay counter 32 is used to inhibit chip operation for a period of time following loading to make random tries an unproductive process. Loading of the key registers causes the "access modification allowed" signal to be true in each chip. The ACP 10 then establishes access by loading the access control memories according to the stored information configuration.

As a sixth step, at subsequent authentication dialog, periodically, according to the user's configuration, the ACP 10 prompts an additional user authentication (reauthentication). In the event of failure, the ACP 10 forces all memory chips to their power on states, thus inhibiting any access to the memories' data by clearing the access control memory 43 and clearing the contents of the key register 31. Now, the operation of the system of FIG. 1 will be described with reference to FIGS. 4 and 5.

First Operations of the Day

FIG. 4 shows in block diagram form, the various modes of operation. Blocks 402 and 401 show the two startup conditions. In block 402, the user inserts the memory card 3 in the previously powered-up host processor 5. In block 401, the user powers up host processor 5 with memory card 3 already installed.

In either of the above startup operations, during block 402, the ACP 10 and its interfaces are initialized in a conventional manner, and block 403 clears all of the 'n' K registers 31 and the 'n' access control memories 43 as part of the flash memories 103a through 103n internal initialization sequence. This prevents any data from being read out of memories 103a through 103n since output buffer 52, in each memory, is disabled. The lock value is loaded into the 'n' L registers 33 from the respective LMs 35 as a result of power on.

Now in block 404, ACP 10 sends an interrupt signal to host processor 5 which responds by requesting the PIN or other identifying information from the user. In block 405, ACP 10, by means of the program stored in memory locations 10-2a, checks that the PIN or other identifying information matches the information stored in memory locations 10-2a. If no match, then decision block 406 counts an error and ACP 10 branches to block 404 to repeat the test. If the test fails a preset number of times, then decision block 406 branches to block 407 to cause ACP 10 to either lock up or destroy the contents of the memories 103a through 103n.

First User Authentication Successful

If in decision block 406 there is a match indicating a successful authentication then in block 408, the ACP 10 via a load key register command loads each K register 31 from memory locations 10-2b with the appropriate key value. Also block 409 repeatedly decrements the contents of delay counter 32 issuing successive the decrement delay counter commands toward a binary zero count which causes the generation of the zero count signal 41 in FIG. 3.

In block 410, each access control memory 43 location is loaded with information by means of the read-allow memory bank command to allow access to the selected banks of the corresponding flash memory 103a through 103n.

Intermittent Re-authentication

In block 411, the ACP 10 awaits the end of the preset time interval established by information stored in memory locations 10-2a signalled by interval counter 10-8 before requesting user re-authentication. Then, in block

412, the ACP 10 interrupts the host processor 5 to request the user to re-enter the PIN or other required identification.

Decision block 413 checks the PIN or other information received from the host processor 5 against the information stored in memory locations 10-2a and the interval timer 10-8 output is recorded. The user has a preset time interval of typically 30 seconds in which to enter the authentication information into host processor 5. While the clock is running, if the decision block 413 test fails, then block 414 records the test as an error. At that time, it checks if a maximum number of errors was received and branches to repeat blocks 412 and 413. If the number of errors equals the maximum number, then in block 415, APC 10 clears the flash memory K register 31 by means of successive load key register commands, and clears the access control memories 43 with successive read-disable memory commands. Block 415 then branches to block 404 to allow a new "First Authentication" operation to take place.

If the test in decision block 413 is successful, the K register 31 remains unchanged (i.e., contains the key value previously loaded by the ACP) enabling the user to continue to operate the system 1. In the event that the 30 seconds elapsed without decision block 413 receiving the PIN or other information, the ACP 10 clears the K register 31 and the access control memory 43 as before.

FIG. 5 is a flow diagram which illustrates how host processor 5 responds to an interrupt request from APC 10 for authentication in response to blocks 404 and 412 of FIG. 4. As shown, decision block 501 is waiting for an interrupt from the ACP 10 requesting that the user re-enter the PIN or other information. Decision block 501 branches to block 502 when it receives the interrupt from blocks 404 or 412. Block 502 displays the request for the PIN or other information on host display 5-2. Block 503 accepts the information from the keyboard and block 504 interrupts ACP 10. Block 5 sends the PIN to ACP 10.

It will be appreciated by those skilled in the art that many changes may be made to the preferred embodiment of the present invention without departing from its teachings. For example, the invention may be used with different types of non-volatile memories and different interfaces, etc.

While in accordance with the provisions and statutes there has been illustrated and described the best form of the invention, certain changes may be made without departing from the spirit of the invention as set forth in the appended claims and that in some cases, certain features of the invention may be used to advantage without a corresponding use of other features.

What is claimed is:

1. A secure memory card for use with a host portable computer, said memory card comprising:

a microprocessor connected for transmitting and receiving address, data and control information to and from said host computer and said microprocessor including:

an addressable non-volatile memory for storing information including a number of key values and configuration information;

an internal bus connected to said microprocessor for transmitting address, data and control information defining memory operations to be performed by said card; and,

at least one non-volatile addressable memory being connected to said internal bus in common with said microprocessor for receiving said address, data and control information, said memory including a non-volatile memory section and a security control section, said memory section containing a memory array organized into a number of blocks, each block having a plurality of addressable locations and control logic means for performing said memory operations and said security control section being connected to said internal bus, to said control logic means and to said memory array, said security control section including:

a number of non-volatile and volatile storage devices for storing at least one of said key values and configuration information associated with said blocks; and,

access control logic means connected to said control logic means and to said storage devices, said access control logic means enabling reading of information stored in addressed ones of said blocks of said memory array as specified by said configuration information only after said microprocessor has determined that a predetermined authentication procedure has been performed with said host computer and has enabled said access control logic means for allowing reading of said information from said memory array according to said configuration information.

2. The memory card of claim 1 wherein said microprocessor and said non-volatile memory are included on separate semiconductor chips.

3. The memory card of claim 1 wherein said card further includes interface circuit means coupling said card to said host computer and wherein said interface circuit means and said microprocessor are included on the same semiconductor chip.

4. The memory card of claim 1 wherein said non-volatile memory and said non-volatile storage devices are flash memories.

5. The memory card of claim 1 wherein one of said non-volatile storage devices is a lock memory for storing a lock value corresponding to said one key values and a second one of said non-volatile devices is a lock storage enable element which connects to said lock memory, said lock memory being initially loaded with said lock value and said lock storage enable element being switched to a state which inhibits modification of said lock value under control of said microprocessor.

6. The memory card of claim 2 wherein storage of said lock value and switching of said lock storage enable element takes place during initial fabrication of said memory card.

7. The memory card of claim 5 wherein one of said volatile storage devices is an addressable access control memory having a plurality of locations corresponding in number to said number of blocks of said memory array for storing said configuration information, said access control memory being connected to said internal bus and to said access control logic means, said access control memory being loaded under control of said microprocessor only after said microprocessor has determined that said predetermined authentication procedure initially has been successfully performed with said host computer causing enabling of said access control memory by said access control logic means.

8. The memory card of claim 7 wherein said lock value loaded into said lock memory is all ONES and wherein said security control section further includes an all ONES detector circuit connected to said lock mem-

ory, said detector circuit in response to said lock value of all ONES generating a signal which effectively bypasses said security control section enabling said non-volatile memory to operate as if said security control section had not been included.

9. The memory card of claim 7 wherein performance of said predetermined authentication procedure initially takes place when said memory card is first connected to communicate with said host computer.

10. The memory card of claim 9 wherein said access control means includes a lock register connected to receive said lock value from said lock memory, a comparator circuit, a key register for storing a key value transferred to said key register by said microprocessor, a delay counter for storing a count defining a predetermined time interval and gating means connected to said access control memory, to said comparator and to said delay counter, said comparator circuit being connected to said lock and key registers and to said gating means and said gating means being connected to said delay counter for generating an access modification allowed signal in response to said comparator circuit signalling an identical comparison between said lock code value loaded into said lock register when said delay counter has signalled an end of said predetermined time interval, said access modification allow signal conditioning said access control memory for loading said configuration information.

11. The memory card of claim 10 wherein said control logic means includes circuits for generating command signals in response to a predetermined set of commands used by said microprocessor in controlling the operation of said security control section of each memory chip.

12. The memory card of claim 11 wherein said control logic means in response to a first one of said predetermined set of commands generated by said microprocessor, generates a first signal for loading said lock code value into said lock memory, said first one of said predetermined commands being generated during initial fabrication of said card.

13. The memory card of claim 12 wherein said control logic means in response to a second one of said predetermined set of commands generated by said microprocessor generates a second signal for switching said lock storage enable element to a predetermined state which inhibits said reading or said modification to said lock value stored in said lock memory.

14. The memory card of claim 12 wherein said control logic means in response to a third one of said predetermined set of commands generated by said microprocessor, generates a third signal for loading said key register with a predetermined one of said key values, said third one of said predetermined set of commands being generated by said microprocessor only after said microprocessor has determined that said predetermined authentication procedure has been successfully performed.

15. The memory card of claim 14 wherein said third signal generated by said control logic means simultaneously forces said delay counter to a predetermined count for establishing a start of said predetermined time interval and wherein said control logic means in response to each fourth one of said predetermined set of commands generated by said microprocessor decrements by one, said predetermined count, said delay counter signalling said end of said time interval following execution of a predetermined number of said fourth ones of said set of predetermined commands.

16. The memory card of claim 11 wherein said control logic means in response to a number of fifth and sixth ones of said predetermined set of commands by said microprocessor, generates fifth and sixth signals for setting and resetting locations in said access control memory according to said configuration information for defining which ones of said blocks from which information is allowed to be read out.

17. A secure memory card installable in a host portable computer for establishing communication with said host computer, said memory card comprising:

- a microprocessor contained on a single semiconductor chip, said microprocessor being connected for transmitting and receiving address, data and control information to and from said host computer and said microprocessor including:
- an addressable non-volatile memory for storing information including a number of key values defining user accessibility to memory areas, and memory configuration information defining memory read out accessibility to said memory areas;
- an internal bus for transmitting address, data and control information defining memory operations to be performed by said card; and,
- at least one non-volatile addressable memory chip being connected to said internal bus in common with said microprocessor for receiving said address, data and control information, said memory chip including a memory section and a security section, said memory section containing a non-volatile memory array having a data output and being organized into a number of blocks, each having a plurality of addressable locations and control logic means for performing said memory operations, said security section being connected to said internal bus, to said control logic means and to said data output and said security section including:
  - a non-volatile lock memory coupled to said internal bus for initially receiving and permanently storing a predetermined lock value which matches one of said number of key values;
  - access control logic means connected to said control logic means and to said lock memory for generating an enabling signal upon detecting when said predetermined lock code value identically matches a selected one of said key values applied by said microprocessor to said internal bus; and,
  - an addressable volatile access control memory having a plurality of locations corresponding in number to said number of blocks of said memory array for storing said memory configuration information defining said read out accessibility, said access control memory being connected to said control logic means, to said memory array data output, to said internal bus, and to said access control logic means, said access control logic means enabling reading of information stored in addressed ones of said blocks of said memory array as specified by said memory configuration information only after said microprocessor has determined that a predetermined authentication procedure has been successfully performed with said host computer and has transferred said predetermined one of said memory key codes causing said access control logic

means to generate said enabling signal for application to said data output for enabling reading out said information to said data output as specified by said access control memory configuration information.

18. A secure memory card including a number of non-volatile memory chips, each memory chip including a memory array organized into blocks of addressable locations, having a capability of operating in a number of modes, said card comprising:

a lock memory for storing a lock value;

control means for generating first and second commands and a predetermined key value;

a key register coupled to said control means and responsive to said first command for storing said predetermined key value;

a comparator coupled to said lock memory and to said key register, said comparator generating a compare signal whenever said lock value and said predetermined key value are equal;

a delay counter coupled to said generating means and responsive to said first command for setting said counter to a maximum count value, and responsive to a sequence of successive second commands for generating a zero count signal when said delay counter has been decremented to zero;

logic circuit means coupled to said comparator and to said delay counter, said logic circuit means responsive to said compare signal and said zero count signal for generating an access modification allowed signal;

said control means for generating a third command, and first address signals and subsequent address signals identifying a first of said blocks and subsequent blocks respectively; and,

access control memory means being coupled to said logic means and to said control means, said access control memory responsive to said access memory enable signal, said address signals and said third command for storing indications signifying when said one of said blocks and said subsequent blocks are enabled for reading.

19. The system of claim 18 wherein said predetermined value and maximum values are selected to be sufficiently large so as to prevent ease of access to said information stored in said non-volatile memory when said memory card is placed in an unauthorized host computer.

20. The card of claim 18 wherein said control means includes a microprocessor which couples to said memory which, upon successfully performing a first user authentication operation, generates said first, second and third commands.

21. The card of claim 20 wherein said first command is a load key command, said second command is a decrementing command and said third command is a read allow block command.

22. The card of claim 18 wherein said memory further includes command control means for decoding a predetermined set of commands for conditioning said card to perform normal memory operations, and said command control means including means for decoding an additional set of commands including said first, second and third commands for providing security for information stored in said memory.

23. A method of organizing for operation, a secure memory card installable in a host computer which includes a number of non-volatile memory chips, each memory chip including a memory array organized into blocks of addressable locations and control logic circuits for generating command signals for performing memory operations, said method comprising the steps of:

(a) incorporating a microprocessor into said card which is connected to communicate with said host computer when installed therein, said microprocessor including an addressable non-volatile memory for storing information including a number of key values defining user accessibility to memory areas and memory configuration information defining accessibility to said memory areas;

(b) incorporating security logic circuits into each non-volatile memory chip, said security logic circuits including a non-volatile lock memory for storing a predetermined lock value, access control logic means connected to said lock memory and an addressable volatile access control memory having a plurality of locations corresponding in number to said number of blocks for storing accessibility bit information according to said configuration information;

(c) interconnecting said microprocessor to each memory chip for transferring address, data and control information to said each memory chip;

(d) modifying said control logic circuits to be responsive to a plurality of commands for operating said security logic circuits;

(e) connecting said microprocessor for performing an initial preestablished user authentication operation with said host computer; and,

(f) connecting said security logic circuits to be enabled by said microprocessor transferring specific ones of said plurality of commands to said each chip only when said authentication operation in step (e) has been successfully performed for allowing said information stored in different ones of said blocks to be read out according to said accessibility bit information stored in said access control memory.

24. The method of claim 23 wherein said microprocessor non-volatile memory has a number of sections and wherein said key values are provided by generating random values for said key values which are to be loaded into a first one of said number of sections.

25. The method of claim 23 wherein said method further includes the steps of: (g) including an interval counter in said microprocessor; (h) connecting said interval counter to said microprocessor non-volatile memory and said interval counter being loaded with a value corresponding to a user selected time interval;

(i) connecting said microprocessor for periodically initiating said user authentication operation of step (e) at said user selected time interval; and,

(j) connecting said security logic circuits to be enabled for continuing to allow said information stored in said blocks to be read out according to said accessibility bit information as long as said authentication operation of step (e) is successfully performed.

26. A method of constructing a secure memory card which includes a number of non-volatile memory chips for storing large quantities of information, each memory chip including a memory array organized into blocks of addressable locations and control logic circuits for generating command signals for performing memory operations, said method comprising the steps of:

(a) incorporating a microprocessor into said card, said microprocessor including an addressable non-volatile memory for storing information including a number of key values defining user accessibility to memory areas and memory configuration information defining accessibility to said memory areas;

(b) incorporating security logic circuits into each non-volatile memory chip, said security logic circuits including a non-volatile lock memory for storing a predetermined lock value, access control logic means connected to said lock memory and an addressable volatile access control memory having a plurality of locations corresponding in number to said number of blocks for storing user accessibility bit information in accordance with said configuration information;

(c) interconnecting said microprocessor to each memory chip for transferring address, data and control information to said each memory chip; and,

(d) modifying said control logic circuits to incorporate a plurality of commands for operating said security logic circuits as an extension to a set of commands normally provided by said control logic circuits whereby said security logic circuits protect said information contained in said number of chips from being read out in an unauthorized manner even when said chips are removed from said memory card.

* * * * *